APPLYING GENETIC ALGORITHM EFFECTIVELY TO THE ASSEMBLY LINE BALANCING PROBLEM

A Thesis Submitted in Partial
Fulfillment of the Requirements
for the Degree of
Master of Technology,
by

Prakhar Gaur

Department of Industrial and Management Engineering

Indian Institute of Technology Kanpur

April, 1995

17-4-95
[illegible]

CERTIFICATE

This is to certify that the present work entitled "*Applying Genetic Algorithm effectively to the Assembly Line Balancing problem*" by *Prakhar Gaur* has been carried out under my supervision and has not been submitted elsewhere for the award of a degree

RKAhuja

April, 1995

(Dr R K Ahuja)
Associate Professor
Industrial & Management Engineering
Indian Institute of Technology
Kanpur - 208 016

IME-1995-M-GAU-APP

ABSTRACT

A number of methods, both heuristic and exact exist in the literature for solving the simple assembly line balancing problem In this thesis, a genetic algorithm has been proposed for the *SALB Type-I problem* Two major types of non-standard, greedy crossover schemes, namely CROSS-1 and CROSS-2 have been presented which forms the backbone of the genetic algorithm

The idea that the species (parent solution in the form of a string of coded numbers) should improve or evolve generation by generation has been incorporated to a greater extent in the crossover scheme, CROSS-2 Owing to its improvisation nature, the algorithm improves a starting solution and hence is termed effective The method of genetic search employed here does not have a random basis but the idea of optimization is fused in it hence the scheme is greedy by nature Named 'EGAALB' (Effective Genetic Algorithm for Assembly line balancing) it obtains an optimal solution for all the standard test problems drawn from the literature, which are upto 111 tasks each When tested to judge the capability of EGAALB on the famous literature problems, it was found that the result of this algorithm is better than all the existing heuristics for SALBP This heuristic is also effective in the sense that it gives optimal or near optimal solution in polynomial time, when the problem of line balancing is known to be of 'NP-Hard' type It is expected that the algorithm will give near optimal balance for tasks up to 1000 or more

A faster variant of EGAALB is also suggested which gives good balance quickly

ACKNOWLEDGMENT

I take this opportunity to convey my heartfelt thanks to Dr R K Ahuja for seing this thesis to a fruitful end Of particular importance was his contribution in bringing the standard data set of problems on Assembly line balancing which could not have been accesed otherwise He gave me an insight into the concept of randomisation which forms the backbone of the heuristics developed in my research work

I would like to extend my sincere thanks to K Srinivas for providing a subroutine which generates a random number between two limiting integers His subroutine was of great use in my computer program for EGAALB I would further thank him for providing good reading material on Genetic algorithms

My thanks are also due to Devendra srivastava for giving me a good research paper on Assembly line balancing using GA Discussions with him have been fruitful and contributive towards my thesis work

The constant encouragement and moral support provided by my batchmates from time to time are worth mentioning And, I would like to thank them from the depths of my heart for making my stay at I I T , a memomorable one

Finally, I am indebted to almighty for helping me by various noticed and unnoticed ways, without which it would have been impossible to bring this thesis in its present form

Date April, 1995 Prakhar Gaur

Place I I T Kanpur ( )

CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

INTRODUCTION

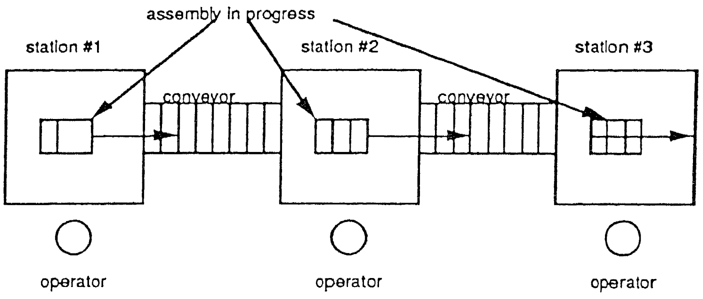
1 1 THE CONCEPT OF PRODUCT ASSEMBLY

A dictionary definition of "assembly" that is suitable to a manufacturing environment is the following to fit or put together the parts of _____ All components that have been fabricated / purchased are combined into a working unit or sub-unit Defective components can halt the assembly function Assembly line balancing is thus an important, pre-assembly micro production planning function, which facilitates the smooth flow of assemblies along a progressive system

1 2 BASIC CONCEPTS OF ASSEMBLY LINES

The basic idea of an assembly line is that a product is progressively assembled as it is transported, past relatively fixed assembly stations, by a material handling device such as a conveyor The *assembly system*, comprising of work stations linked together by a transport mechanism, performs a set of distinct *minimum rational work elements (tasks)* for the assembly of a product A *task* is the smallest indivisible work element and a *workstation* is a location where the tasks are processed A station consists of human or robotic operators and /or machinery, equipment and gadgets The series of stations and conveyor is more commonly known as *assembly line* Exhibit (1 1) shows a layout template of a typical assembly line

A manufacturing item is fed to the first station of the line at a pre-determined constant feed rate The conveyor moves at each interval of *T* time units, known as *cycle time* The time required for the completion of a task is termed the *process time* The sum



Schematic of Balanced 3 Station System

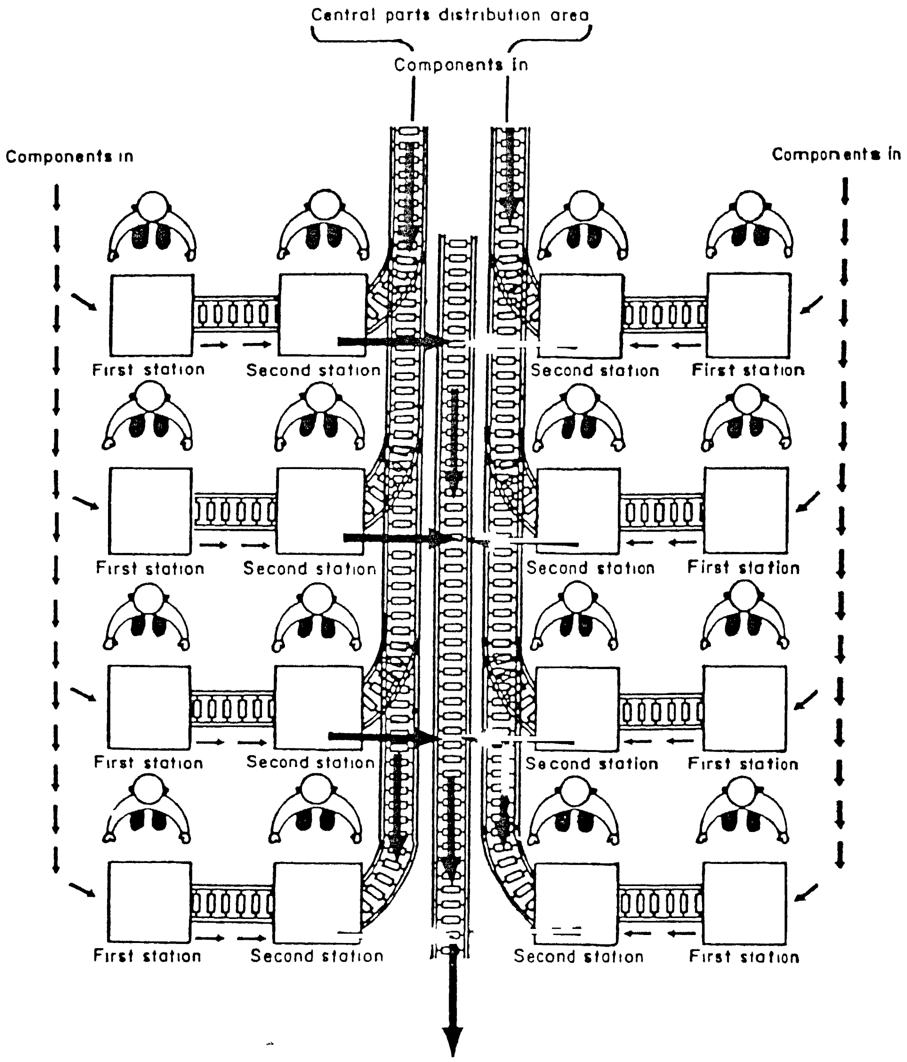


Figure 1 1 Layout template of a typical assembly line

of the process times of all tasks assigned to a station for processing is known as the *work content* of that station Since the item is available to a station only for T units (cycle time), the work content of a station should not exceed T, so that the line can operate smoothly without delays The *production rate (or feed rate)* for the system is $1/T$, i e , one unit of the finished product emerges from the last station along the line every T units of time The cycle time is traditionally pre-determined, based on the demand for the product in the given period and/or the given operating time for the manufacturing system in that period

The tasks can not be assigned to the stations arbitrarily because of technological sequencing requirements, known as *precedence relations* The precedence relations are represented schematically by a *precedence matrix* or by a *precedence network diagram* whose nodes correspond to the tasks

1 3 THE IDEA OF BALANCING AN ASSEMBLY LINE

In addition to the precedence relations there may be more elaborate restrictions, such as zoning constraints which prevent grouping certain tasks at the same work station For maximum resource utilization, the collection of tasks assigned to a particular station must be chosen so that the idle time at that station is minimized The assembly line is said to be *balanced* if total slack (i e , the sum of the idle times of all the stations along the line) is as low as possible For a fixed cycle time, this can be achieved by minimizing the number of stations The line is said to have *perfect balance* if the tasks can be grouped so that all the station work contents are equal However, in real life situations, it is very difficult to achieve a perfect balance

To distinguish from assembly lines, *transfer line* is an automated flow line with automatic material handling from station to station and with specialized task processing

The tasks performed in transfer lines are often restricted by fixed machine cycles (Groover, 1980), unlike in assembly lines where the main restriction is production rate

1 4 VARIATIONS OF THE ALB PROBLEM

Generally, the assembly line balancing problem and methodologies can be categorized as

(i) Mechanically paced production line In such a model the processing time at each work station is constant and the units are advanced mechanically through successive stages Such a model is of *deterministic* type, i e , all input parameters are assumed to be known with certainty

(ii) Stochastic production line When the processing time at each station is variable, specified by a statistical distribution, we have a series of queuing systems The alternating conditions of congestion and idleness for each station makes it difficult to achieve a balanced production line The allowance of 'in process inventory' may reduce the unevenness in the overall output rate but does not solve the balance problem completely

(iii) Mixed production line A combination of the first two types is quite prevalent in practice when large segments of production systems are considered

Even though the majority of the actual systems are of stochastic type, waiting line theory has not been of significant value in solving the line balancing problem Most methods advanced to date address themselves to the simplified case of a mechanically paced line for which service times are constant and units are always available in the form of 'in process inventory

The algorithms available for the assembly line balancing problem are of two types, i e *exact* (or optimum seeking) algorithms and the *inexact* (heuristic or approximate) methods

In contrast to the stochastic line balancing problem the deterministic line balancing problem is referenced in literature with its two types called *type-1* and *type-2 problem* The objective of type-1 problem is to minimize the number of stations or equivalently minimize the total idle time for a given cycle-time Whereas, the type-2 problem aims to minimize the cycle time for a given number of work stations

The deterministic problem is also categorized as simple assembly line balancing problem and generalized ALBP SALBP is termed "simple" in the sense that no "mixed models", "sub assembly lines", "zoning restrictions" etc are mixed

1 5 OUTLINE OF THIS RESEARCH WORK

The first chapter serves as an introduction to the problem of assembly line balancing and its variations A literature survey of the existing solution techniques is done in the second chapter It also covers the mathematical programming formulation of the assembly line balancing problem

The philosophy and methodology of genetic algorithm, in general has been described in the third chapter Genetic algorithms, which is a relatively recent optimization technique, has been employed to solve the SALBP-1 Several heuristic and crossover schemes have been developed and are presented in Chapter-4 The heuristics were coded in 'C' programming language and were tested on standard literature problems

In the fifth chapter, the results are compared with several heuristics, existing and developed by researchers in the past It was found that one of the heuristic developed in this research work is superior than all of the heuristics existing in literature, when tested on the classic literature problems No other heuristic gave optimal answers for all the test problems

Chapter 2

ASSEMBLY LINE BALANCING

2 1 HISTORY OF ALB TECHNIQUES

Salveson (1970) studied the computational performance of heuristics to solve the assembly line balancing problem The problem of ALB was first defined analytically in the open literature by Salveson (1955) Since then, many algorithms and heuristics have been presented in the literature to solve Salveson's problem, and many other formulations, representing complications of Salveson's model Mastor (1970) studied computational performance of heuristics to solve Salveson's model

Jackson (1956) was first to present an algorithm of any practicality, since Salveson had presented a linear programming model which could result in split stations Although not formulated in today's usual dynamic programming terminology, it was well tailored to fit the ALBP He presented optional refinements to his basic formulations later

Held & Karp (1962) presented another dynamic programming formulation, which they refined with Sharesian (Held et al, 1963) Their formulation is based on feasible sets of tasks Each set has a 'cost' being time committed to the tasks in the set, with consideration to the station allocation The algorithm minimizes cost, and therefore the required number of stations

Gutjahr & Nemhauser (1964) presented an algorithm which utilized the same concept of feasible sets as Held et al , but searched them without resorting to dynamic programming Conway (1967) pointed out that the branch and bound methods could be applied to ALB problems Charlton & Death (1969) defined a branch and bound algorithm where each arc represented the assignment of a single task, in frontier search Johnson

(1973) defined a branch and bound algorithm where each arc represented a station in a newest node search An improvement of this algorithm was presented by Johnson himself in 1981

The three major categories of assembly lines are - (a) Stochastic, (b) deterministic and (c) mixed model lines

2 1 1 Stochastic Line Balancing

The traditional approach to the line balancing problem assumes that the task times are deterministic However, tasks requiring human skills usually exhibit noticeable variations in task times, resulting in occasional incompletions of the assigned tasks Moodie & Young (1965) are the first to have considered explicitly the stochasticity of task times, and they presented a procedure for assigning tasks to stations in such a way that the probability of completing the tasks assigned to any stations is no less than a prescribed level (say 0 9) Later, Reeve & Thomas (1973) investigated the effectiveness of four different procedures for reassigning tasks to stations of an initial line design in order to minimize the probability of one or more stations exceeding the cycle time The studies assumed that in the final line designs, the probability of completing all the tasks will be sufficiently high, and therefore the actual occurrence of incompletions and their associated costs and consequences may be ignored

However, Kottas & Lau (1976) have shown that, for many line designs, the cycle time (hence the labour cost) will have to be uneconomically high before the occurrence of incompletions will become negligible A meaningful stochastic line balancing procedure should produce a line design that minimizes a cost function with two components - (i) the incompletion cost component, and (ii) the labour cost component Kottas & Lau (1973) presented a heuristic line design procedure that adopts such a dual component objective function , but their heuristic is a deterministic one that produces only one line design , and no alternative is available if the heuristic line design turns out to be unsatisfactory

The second paper of Kottas & Lau (1976) presented a valid cost model and an efficient computational procedure for determining the sum of the labour cost and incompletion cost components of any given line design The third paper by Kottas & Lau (1981) presents a comprehensive stochastic line balancing procedure with two basic stages , the first stage is a 'probabilistic' heuristic that will generate many promising low cost line design for any given line balancing problem in the second stage the earlier cost model (1976) is used to evaluate those generated line designs and to identify the most economical ones

2 1 2 Mix Model Lines

Model mix assembly lines are concerned with the progressive assembly on a simple line of several models of a particular product type, e g , vehicles, refrigerators Whilst ensuring that common (or similar) activities required for each model are performed by the same operator, the main problem arises in determining the sequence in which the models are assembled

The sequencing problem with model-mix assembly lines was first defined by Kilbridge & Wester (1963) The goal of the methods in mixed model lines is to determine the sequence of models that will balance the work load of the station over the different models If the work station is such that the worker is mobile (walks a short distance while working on the unit as it moves), a good sequencing method will minimize the number of extreme movements required by an operator between successive units on the line

Methods proposed by Thomopoulos, Dar-El & Clothers and others work toward improving on this criterion

2 2 HEURISTIC LINE BALANCING TECHNIQUES

Mastor (1966) evaluated the performance of 10 heuristic decision rules on 20 and 40 task assembly problems for 3 different order strength (ratio of the number of ordering relations among tasks to be grouped into work stations to the total no of possible ordering), and for different line lengths (number of work stations) Mastor's general conclusion is that the best results are obtained with Held et al (1963) dynamic programming technique The results obtained with Held et al dynamic programming technique were followed closely by Arcus' COMSOAL (1963) in Mastor's investigation

Dar El Mansoor (1975) investigated 12 heuristic decision rules, (for type II problems) and conclude that MALB, a technique he developed (1973), which is based upon backtracking extension to the Ranked Positional Weight heuristic of Helgeson & Bernie (1961), gives consistently superior results to the Arcus or to the other techniques investigated

The heuristic decision rules developed by the researchers can be categorized in four major categories which are as follows

2 2 1 Single Pass Decision Rules

The first category consists of those decision rules which implement a list prioritizing scheme for task assignments based upon single attribute of each assembly task Operationally, a task is first assigned a numerical priority specified by the logic of the heuristic decision rule Then tasks that are both precedence and cycle-time feasible (i e , all predecessors have been assigned to a work station and the task time is not larger than the remaining time available at the work station) are placed on an available list The task on the available list with the highest priority is assigned first The available list is then updated to reflect the possible addition of tasks that are now precedence feasible, and the amount of time available to be assigned to tasks in the work station is reduced by the task time of the assigned task This process continues for a station until no more tasks can be assigned to it The assignment process then continues to the next station, and so on, until all tasks have been assigned to some work station When the final task has been assigned a complete balance has been obtained Table 2 1 summarizes various such decision rules

2 2 2 Composite decision rules

This category consists of decision rules which are a composite of single pass decision rules or otherwise produce multiple single pass solutions for a given problem, selecting that solution for implementation which results in fewest number of stations One composite rule, COMPOSITE-13 selects the best solution available from Category I decision rules

The Arcus' (1963) biased sampling procedure generates feasible sequences of tasks for assignment to a work station A "fit list", consisting of those tasks which can be assigned to a work station is constructed, and weights governing the probability the task will be selected for assignment to the work station are assigned to each task Tasks so assigned are removed from the fit list, and a new fit list, consisting of the tasks which can currently be assigned to some work station A given line is then balanced several times, resulting in different assignments based upon the probabilistic selection of tasks from the fit list To bias the selection of a task into a station, Arcus' Rule IX uses a product of five separate weights

2 2 3 Backtracking Decision Rules

The approaches under this category attempts to improve upon a solution or station assignment previously obtained Hoffmann (1963) developed a heuristic method Starting with station one, a precedent feasible list of tasks is maintained from which the combination of tasks which will minimize station idle time is found via complete enumeration This procedure is repeated for each station in numerical order, until all tasks have been assigned Gehrlein & Patterson (1975,1978) have shown that slight modification to Hoffmann's original procedure can have significant impact on reducing the amount of time devoted to each search while simultaneously smoothing the idle time present among work stations for certain classes of line balancing problems

Dar-El developed MALB (1973) as a heuristic variant of his earlier optimal seeking iterative procedure (1964) His optimal seeking procedure is based upon the Rank Positional Weight heuristic method of Helgeson & Bernie (1961) enhanced with a backtracking algorithm that generates all feasible sequences of task assignments Dar-El found that the computation time of his optimal seeking method restricted its applicability, so he switched over to a heuristic approach which retains the power of optimal approach without taking extra computational time

2 2 4 Optimal Seeking Decision Rules

Clearly, given enough computation time, an optimal seeking procedure will dominate any of the heuristic methods when optimality of the solution is the criterion Magazine & Wee (1981b) report excellent results for the type I line balancing problem with their branch and bound procedure The branching direction, fathoming criteria, and growth rate of the tree are controlled via a number of heuristics and dominance tests Two decision rules of special significance used are IUFFD (Immediate Update First Fit

Decreasing) which is referred to as MAXDUR, and IUBRPW (Immediate Update Backward Recursive Positional Weight)

Talbot & Patterson (1984) gave a depth-first, implicit enumeration, backtracking, integer programming procedure to which various search, fathoming and backtracking rules are applied The first variation ALBCUT contains network cuts (1984), where as second variation ALBHOFF uses MAXDUR etc for search and backtracking

Schrage & Baker (1978) proposed an efficient method for implementing the dynamic programming approach of Held et al (1963) through improved procedures for generating feasible subsets Magazine & Wee concluded that branch and bound methods are superior to dynamic programming with regard to computation time and computer storage required

Multiple Solutions Technique, MUST by Dar-El & Rubinovitch (1979) employs exhaustive enumeration to generate all solutions, or some subset of them They demonstrated that MUST dominates MALB These methods of Section 2 2 4 are exact procedures but can be considered as heuristic when a constraint on CPU time is implemented

The performance of several heuristics is compared in Fig (5 3) by plotting the average percent increase above optimal solution for standard literature problems The figure shows that the backtracking heuristics, and specially Hoffmann heuristics are the best amongst all

2 3 THE DETERMINISTIC ASSEMBLY LINE BALANCING PROBLEM

The following are the assumptions for the ALB model discussed in this chapter

(A-1) All input parameters are known with certainty

(A-2) A task can not be split among two or more stations

(A-3) Tasks can not be processed in arbitrary sequences due to technological precedence requirements

(A-4) All tasks must be processed

The ALB problem is assigning the tasks to the stations while optimizing some criterion and not violating a number of possible restrictions or requirements which will be discussed below

2 3 1 The Simple ALB Problem (SALBP)

In addition to (A-1) - (A-4) above, the following are true for SALBP

(A-5) All stations under consideration are equipped and manned to process any one of the asks (i e , it is assumed in effect, that the fixed and variable costs associated with all the stations are the same and, therefore, they need not be considered in the model)

(A-6) The task process times are independent of the station at which they are performed and of the preceding or following tasks (i e , process times are fixed and furthermore, they are not sequence dependent)

(A-7) Any task can be processed at any station (i e , there are no positional, layout or zoning restrictions)

(A-8) The total line is considered to be serial with no feeder or parallel sub-assembly lines (and, therefore, process times are additive at any station) or any possible interactions of this type is ignored

(A-9) The assembly system is assumed to be designed for a unique model of a single product

We can now define the first version, namely SALBP-1, of the SALBP In addition to (A-1)-(A-9), we have

(A-10) The cycle time T is given and fixed

The goal is to *minimize total slack* which is equivalent to *minimizing the number of stations* along the line (see Fig 1 1) SALBP-1 is also referred to as the *"line balancing problem"* by Jackson (1956) and Freeman (1968) or the *"single model ALBP"* by Dar-El (1975) and Pinto Dannenbring & Khumawala (1978) or *"Type -1 ALBP"* by Mastor (1970) and Wee & Magazine (1981) or *"the basic ALBP"* by Johnson (1983)

The second version of the problem, namely SALBP-2, is the same as SALBP-1 except that instead of (A-10), we have -

(A-11) The number of stations is given and fixed

The goal is to *minimize the cycle time* or equivalently, to *maximize the production rate* SALBP-2 is also known as the *"Type -2 ALBP"* in the research works of Mastor (1970) and Wee & Magazine (1981)

2 3 2 The General ALB Problem (GALBP)

ALBP has been defined in many different ways by relaxing one, or any combination, of the assumptions (A-1)-(A-9), with the exception of (A-5) which will be discussed later The line may be used for the production of two or more models of the same product in batches, known as *multi - model* case or the line may be used for the production of two or more models of the same product, not in batches but they may be intermixed, known as *mixed - model* case (e g Thomopulos 1967, Dar-El 1978 and Dar-El & Cother 1975) Furthermore, there may exist *zoning constraints* restricting the grouping of certain tasks at the same station/ area (e g Mitchell 1957 and Tonge 1960)

There may be restrictions on balance delay (e g Kilbridge & Wester 1961), namely, the amount of idle time on the line due to unequal task assignments to the stations, there may exist *parallel stations* (e g Tonge 1961, and Pinto, Dannenbring & Khumawala 1975) there may exist other *positional restrictions* , buffer stocks and other generalities such as feeder *parallel assembly lines* (e g Nanda & Scher 1976) , other extensions and other generalities in Mansoor (1964) and Freeman (1967) , and a goal programming approach in Gunther, Johnson & Peterson (1983) An excellent discussion of these can be found in Groover (1980) and solution methods for certain cases in Johnson (1983) Whether the goal is to minimize total slack or to minimize the number of the stations along the line, these problems will be referred to as the GALBP or the generalized assembly line balancing problems Thus GALBP is a generalization of SALBP-1 and SALBP-2

In GALBP there is no explicit concern for the cost of stations (fixed cost) and the cost of operating the stations variable cost), (A-5) is true for GALBP (and, therefore, SALBP) Some of these issues have been addressed by Graves & Lamar (1983), for instance, define a station selection and task assignment problem for automated assembly systems in which one of the optimization criteria is "cost" Whereas, Pinto, Dannenbring & Khumawala (1983) deal with the problem of choosing among process alternative, as well as the assignment of the tasks to the stations, while minimizing labour and fixed costs The problem in which (A-5) is not true will be referred to as the ALDP, i e , the Assembly line design problem Thus the main difference between GALBP and ALDP is that, in the latter, the choice problem includes technology/ labour, based on fixed and variable costs (i e , ALDP is a generalization of ALDP) The ALDP models and methods will not be reviewed here

2 4 SALBP AND OTHER RELATED COMBINATORIC PROBLEMS

In terms of practical problems, SALBP falls into the general class of *sequencing* and *scheduling* problems (Baker, 1974) Sequencing problems are scheduling problems in which an ordering of the jobs/ task completely determines the schedule These scheduling problems are further complicated by the fact that the scheduling decisions are generally subject to 'precedence' constraints as well The simplest example in this class is the *single resource scheduling problem,* with resource being a machine processor This problem is closely related to ALBP for instance, if the precedence constraints in SALBP-1 are replaced by the requirement that the immediate predecessor of each task must be assigned to an earlier station, the resulting problem becomes a *single resource constrained scheduling problem* (e g , Garey, Graham, Johnson & Yao 1976) Another special case of SALBP-1 that generated a lot of interest is known as *bin packing problem* which simply is SALBP-1 without precedence constraints a given collection of items (tasks) are to be packed into (assigned to) a minimum number of bins (stations) each with identical finite capacities (cycle time) (e g , see Garey & Johnson 1981) Hence, SALBP-1 is a generalization of the bin packing problem (e g , see Wee & Magazine 1982)

A well known problem related to the bin packing problem, and hence to SALBP-1, is the *knapsack problem* There is now only one bin (knapsack) and only a subset of the given collection of items can be packed into that bin Those items are to be chosen for packing while optimizing some criterion (e g , Balas & Zemel 1980) And finally, there is a well known *partition problem* partition a given collection of objects into a number of disjoint subsets while optimizing some prespecified criterion The subsets of the task set designated for the respective stations is clearly a partition of the task set Hence SALBP is a reduction of the partition problem (Karp 1972)

Since the above problems are clearly related to each other it is possible to incorporate the solution methodology for one problem as a subroutine in the solution methodology for related problem For instance, Talbot & Patterson (1984) repeatedly solved knapsack problems to assist in the solutions of SALBP-1 by eliminating inferior task assignments from consideration Wee & Magazine (1981) use bin packing heuristics to obtain lower bounds for solving SALBP-1

2 5 MATHEMATICAL FORMULATION OF SALBP-1

Salveson (1955) formulated SALBP-1 as a linear programming problem encompassing all possible combinations of station assignments His model, by definition, can result in split task and, therefore, may result in infeasible solutions Bowman (1960) was first to provide a 'non-divisibility' constraint, by changing LP formulation to one of integer programming In the Bowman model, as modified by White (1961), presented here I denotes the task set, $I = \{1,2, \quad ,i, \quad ,m\}$ and t_i the process time of task i, where $i \in I = \{1,2, \quad ,m\}$

Also $P(i)$ are {immediate predecessors of task i} and the cycle-time is denoted by T The decision variable is defined as follows let $x_{ij} = 1$ if task i is assigned to station j and $x_{ij} = 0$ if it is not, where $j = 1,2, \quad , n$, and $n \leq m$ The following 0-1 program represents SALBP-1

Minimize $Z = \sum_{j=1}^{n} \sum_{i=1}^{m} c_j x_{ij}$ (1)

subject to $\sum_{j=1}^{n} x_{ij} = 1 \quad \forall\, i \in I$ (2)

$$\sum_{i=1}^{m} t_i x_{ij} \leq T \quad \forall\, j \in J \tag{3}$$

$$x_{ik} \leq \sum_{j=1}^{k} x_{k} \quad \forall\, i \in I, \forall\, k \in J \text{ and } \forall\, h \in P(i) \tag{4}$$

$$x_{ij} = 0,1 \quad \forall\, i \in I \text{ and } \forall\, j \in J \tag{5}$$

where c_j is the penalty cost associated with using work station j such that

$$c_{j+1} \geq M c_j \quad \forall\, j \in J - \{n\} \tag{6}$$

and M is a sufficiently large positive integer

The objective function (1) represents the 'cost' of using the stations and is, therefore, to be minimized Relation (6) defines such a premium on using an additional station when all the task can be accommodated without it, that the minimization of the objective function (1) will result in the minimum number of stations Constraint (2), known as the *'occurrence constraint'*, guarantees that every task is assigned to a station and the *'cycle time constraint'* (3) guarantees that the total process time for all the tasks assigned to each station is, at most the pre specified cycle time

Finally, constraint (4) represents the precedence relations if $x_{ik} = 1$, i e if task i is assigned to a station k then, each immediate predecessor h of task i must be assigned to a station j such that $j \leq k$, so that task h may be completed before the processing of task i starts If, on the other hand $x_{ik} = 0$, the immediate predecessors h of i may or may not be assigned to the first k stations Constraint (5) of course guarantee that each variable can assume values 0 or 1 only (i e , a task can not be split among two or more stations)

An improved formulation has been suggested by Patterson & Albracht (1975) Talbot & Patterson (1984) have proposed the formulation as a general mixed integer programming problem without using binary decision variables The decision variable in this formulation is defined to be the " Number of station that task i is assigned to", denoted by x_i $\forall$ $i \in I$ Program -TP below is the Talbot and Patterson formulation

$$\text{minimize} \qquad Z = x_m$$

$$\text{Subject to} \qquad \sum_{i \in I(j)} x_{ij} \leq T \qquad \forall\ j \in J,$$

$$x_h \leq x_i \qquad \forall\ h \in P(i) \text{ and } \forall\ i \in I,$$

$$x_i \geq 0 \text{ and integer} \qquad \forall\ i \in I,$$

where m is the unique terminal task, dummy if necessry Clearly, the main advantage of this formulation is that the number of variables equals the number of tasks Furthermore, the objective function has only one term namely the station number of the last (dummy) task in the precedence network

2 5 STATE OF THE ART

The techniques to attempt on Assembly line balancing problems started appearing right from the mid fifties Better and better methods have been received as time passed Johnson, R U (1988) came up with an algorithm 'FABLE' for optimally balancing large assembly lines FABLE is a depth first, branch and bound algorithm with the logic designed for very fast achievement of feasibility and optimality for any line of 1000 or even more tasks

Hackmann, Magazine & Wee (1989)suggested a number of very fast and simple heuristics based on some indexing (weights) criterion This method is very good and easy for small scale problems They also proposed a branch & bound algorithm, which has excellent results due, in large parts, to the repeated use of these indexing heuristics

Hoffmann (1991)proposed a branch and bound algorithm which is superior to the well known Hoffmann (1961) heuristic It introduces a simple bounding rule which uses the concept of the 'theoretical minimum slack time' to achieve rapid solutions The newer version combines the Hoffmann(1961) heuristic to develop a more effective system for solving SALBP problem This is supposedly the best heuristic so far when considering the quality of solution

Very recently, the trend is to apply genetic algorithms to solve the assembly line balancing problem Anderson & Ferris (1994) have used GA to provide good (if not optimal always) solutions for a number of ALB problems They used Arcus' COMSOAL to generate the initial population for their genetic algorithm In a significant proportion of cases the initial set of Arcus contains at least one which is never improved by the GA And in the other cases the best of Arcus solutions is never far from the best solutions found by the GA

The present thesis work is an attempt to develop a Genetic algorithm better than the earlier researchers to solve the same SALBP

Chapter 3

WHAT IS A GENETIC ALGORITHM ?

3 1 GENERAL

Genetic algorithms were invented to mimic some of the processes observed in natural evolution Biologists have been intrigued with the mechanics of evolution since the evolutionary theory of biological change gained acceptance Evolution takes place on *chromosomes* - organic devices for encoding the structure of living beings A living being is created partly through a process of *decoding* chromosomes Evolution is a process that operates on chromosomes Processes of nature (or natural selection) cause those chromosomes that encode successful structures to reproduce more often than those that do not

The process of reproduction is the point at which evolution takes place Mutations may cause the chromosomes of biological children to be different from those of their parents, and recombination processes may create quite different chromosomes in the children by combining material from the chromosomes of two parent

John Holland (1970's) believed that these features if appropriately incorporated in a computer algorithm can yield a technique for solving difficult problems in the way nature has done, - i e through evolution He developed algorithms that manipulated strings of binary digits - 1's and 0's - that he called *chromosomes* Like nature his algorithms solved the problem of finding good chromosomes by manipulating the material in the chromosomes blindly Like nature, they knew nothing about the type of problem they were solving The only information they were given was an evaluation of each chromosome they produced, and their only use of that evaluation was to bias the selection of chromosomes so that those with the best evaluations tended to reproduce more often than those with bad evaluations (or fitness)

These algorithms, using simple encoding and reproduction mechanisms displayed complicated behaviour, and they turned out to solve some extremely difficult problems This methodology can evolve better designs, find better schedules, and produce better solutions to a variety of other important problems that we cannot solve as well using other techniques In reference to its origins in the study of genetics Holland named the field *Genetic Algorithms*

3 2 WORKING METHODOLOGY OF GENETIC ALGORITHM

To begin with let us consider the mechanisms that link a genetic algorithm to the problem it is solving There are two such mechanisms — a way of *encoding* solutions to the problem on chromosomes, and an *evaluation function* that returns a measurement of the worth of any chromosome in the context of the problem Let us discuss in more detail

Most of the times encoding is carried out using bit strings This string is a potential solution to the problem and is referred to as an *individual* A set of various such strings or *members* is called a *population* The population changes over time, but always has the same size, say, *N* Each individual is represented by a single string of characters

An *evaluation function* takes a chromosome as input and returns a number or list of numbers that is a measure of the chromosome's performance on the problem to be solved The interaction of an individual with its environment (problem environment) provides a measure of *fitness,* and the interaction of a chromosome with an evaluation function provides a measure of fitness that the GA uses when carrying out reproduction

Evaluation function plays the same role in genetic algorithm plays in natural evolution The over all concept of GA is *the survival of the fittest*

At every iteration of the algorithm, a *fitness value*, f_i (i = 1, N), is calculated for each of the current individuals Based on this fitness function (or evaluation function) a number of individuals are selected as potential parents These form what is called a *mating pool* The mating pool will have *N* members but some will be duplicate copies of other, so that it contains several copies of some individuals in the current population and no copies of others Individuals which are not selected for the mating pool are lost

Two new individuals can be obtained from two individuals in the mating pool by choosing a random point along the string splitting both strings at that point and then joining the front part of one parent to the back part of the other parent and vice versa Thus parents A-B-C-A-B-C-A-B-C and A-A-B-B-C-C-C-B-A might produce offspring A-B-C-B-C-C-C-B-A and A-A-B-A-B-C-A-B-C when mated This process is called *cross over* It is not necessary for every member of the mating pool to be involved in cross over , some will be left unchanged

Individuals in the mating pool may also change through random *mutation*, when characters within a string are changed directly Normally, each character is given such a small probability of being changed that most of the time individuals are left unaltered The process of crossover and mutation are collectively referred to as recombination The end result is a new population called *the new generation*, and the whole process repeats

Over time this leads to convergence within a population with fewer and fewer differences between the individuals When a genetic algorithm works well the population converges to a good solution of the underlying problem, and the best individual after many generations is likely to be close to the global optimum A model algorithm is given in Exhibit (3 1)

The Genetic Algorithm

1 Initialise a population of chromosomes

2 Evaluate each chromosome in the population

3 Create new chromosomes by mating current chromosomes Apply mutation and recombination as the parent chromosomes mate

4 Delete members of the population to make room for the new chromosomes

5 Evaluate the new chromosomes and insert them into the population

6 If time is up stop and return the best chromosome , if not go to 3

Exhibit 3 1 Top level description of a genetic algorithm

It is important to realize that one simple method of implementing a genetic algorithm (Goldberg 1989) is described here There are many other versions which have been suggested, some of which use different mechanism for reproduction However the central idea is that, if all goes well then, an initial population of unexceptional chromosomes will improve as parents are replaced by better and better children The best individual in the final generation population can be a highly evolved solution to the problem

A genetic algorithm in general is composed of three principal parts – an Evaluation module, a Population module and a Reproduction module

3 2 1 THE EVALUATION MODULE

The evaluation module contains an evaluation function Binary f6, a mathematical evaluation function decodes a chromosome that is a string of 44 bits by converting it into two real numbers it plugs those real numbers into a mathematical function , and it returns the value of the function for those two numbers This value that is returned will be the evaluation of the chromosome The functions should be designed properly in order to *optimize* the results Binary f6, e g is as follows

$$0\,5 - \frac{(\sin\sqrt{x^2+y^2})^2 - 0\,5}{(1\,0 + 0\,001(x^2+y^2))^2}$$

where x and y are two real numbers decoded from the genetic material of the member

3 2 2 The Population and Reproduction module

This module contains a population of chromosomes and techniques for creating and manipulating the module The reproduction module contains techniques for creating new chromosomes during reproduction

3 3 PARENT SELECTION, MUTATION AND CROSS OVER

Several components of GA require further discussion

3 3 1 Roulette Wheel Parent Selection

The purpose of parent selection in a genetic algorithm is to give more reproductive chances, on the whole, to those population members that are most fit One commonly used technique is Roulette Wheel selection and can be described in the following steps -

(1) Sum the fitness of all the population members to find the total fitness

(2) Generate n, a random number between 0 and total fitness

(3) Return the first population member whose fitness added to the fitness of the preceding population members is greater than or equal to n

The effect of roulette wheel parent selection is to return a randomly selected parent Although this selection procedure is random, each parent's chance being selected is directly proportional to its fitness Over a number of *generations* this algorithm will drive out the least fit members and contribute to the spread of the genetic material in the fittest population members Of course it is possible that the worst population member could be selected by this algorithm each time it is used Such an occurrence would inhibit the performance of a genetic algorithm using this selection technique, but the odds of this happening in a population of any size are negligible

This algorithm can be viewed as allocating pie-shaped slices on a roulette wheel to population members with each slice proportional to the member's fitness Selection of a population member to a parent can be viewed as a spin of the wheel, with the winning population member being the one in whose slice the roulette spinner ends up

This technique of parent selection directly promotes reproduction of the fittest population members by biasing each member's chances of selection in accord with its evaluation

3 3 2 One Point Crossover

There are processes that alter chromosomes during reproduction to have children differing from the parents These processes are *crossover* and *mutation* In nature, crossover occurs when two parents exchange parts of their corresponding chromosomes In genetic algorithm, crossover recombines the genetic material in two parent chromosomes to make two children There are several cross operators available in the literature of which the most important is *one point-crossover*

One-point crossover occurs when parts of two parent chromosome are swapped after a randomly selected point, creating two children The mechanism is shown below -

Parent 1	1 0 1 1 0 1		Child 1 1 0 1 1 0 0
		$\Rightarrow$	
Parent 2	0 0 1 1 0 0		Child 2 0 0 1 1 0 1

One important feature of one point crossover is that it can produce children that are radically different from their parents Another feature is that one point crossover will not introduce differences for bit in a position where both parents have same value or same genetic encoding When two parents are selected for mating, then there is a probability p_c, with which the crossover can occur If the crossover doesn't occur then a copy of parents is passed to the new generation as children

Crossover is an extremely important component of a genetic algorithm Many genetic algorithm practitioners believe that if we delete the crossover operators from a genetic algorithm, the result is no longer a genetic algorithm In fact, the use of crossover operator distinguishes genetic algorithms from all other optimization algorithms

3 3 3 Bit Mutation operator

Genetic algorithm s use mutation to create children that differ from their parents, especially when crossover operator fails to do so (in case when both the parents are identical) When bit mutation is applied to a bit string it sweeps down the list of bits, replacing each by randomly selected bit if a probability test is passed Bit mutation has an associated probability p_m, which is typically very low of order say 0 008 The mechanism of bit mutation is demonstrated below -

Old Chromosomes	Random Numbers	New Bit	New Chromosomes
1 0 1 0	801 102 266 373	-	1 0 1 0
1 1 0 0	120 096 005 840	0	1 1 0 0
0 0 1 1	760 473 894 001	1	0 0 1 1

Hence no effective change has been made by the mutation operator for the first two chromosomes

Some genetic algorithm practitioners use bit mutation to flip bits Using this variant, if the probability test is passed, we replace a 1 by a 0, and a 0 by a 1

The development of crossover and mutation schemes is very much problem dependent The success of GA lies in the heart of these operators or schemes When a population consists primarily of similar individuals, we say that it has *converged* To avoid what is called *pre-mature convergence* a genetic algorithm should be carefully designed Genetic algorithms have been successfully applied to a large number of engineering problems The present thesis work is an attempt to use the strength of GA to solve the Simple Assembly Line Balancing Problem, SALBP-1

Chapter 4

GENETIC ALGORITHMS FOR SALBP

4 1 GENERAL

Algonthms based on genetic ideas were first used to solve optimization problems more than 20 years ago (J D Bagley, 1967) following the development of the fundamental ideas of genetic algorithm by John Holland at the University of Michigan An outline description of the way in which a genetic algorithm works is given in the previous chapter

4 2 ANDERSON'S GA APPLIED TO ALBP

The assembly line balancing problem has attracted the attention of many researchers Both heuristic and exact methods have been proposed for its solution Anderson & Ferris (1994) say that they do not expect a genetic algorithm to be as effective as some special purpose heuristics for ALBP They have not demonstrated the superiority of genetic algorithm but have tried to give some indication of the potential of this technique The genetic algorithm called EGAALB developed in the present research work, however is an attempt to demonstrate the superiority as well, apart from showing the applicability of GA to SALBP

4 2 1 Coding

In this coding technique, a solution is represented by a string, each element of which is a number The number in the *i*th place in the string is the station to which the *i*th task is to be assigned For the example of Fig 4 1, the assignment would be coded as

1 2 2 1 2 2 3 2 3 3

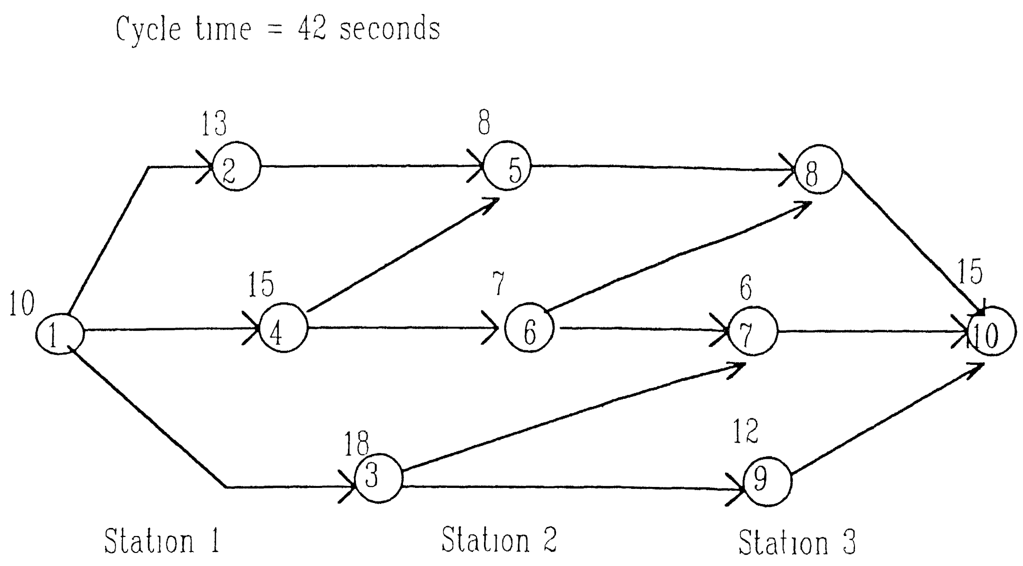


Figure 4 1 An example of precedence network

4 2 2 Achieving feasible solutions

If tasks are assigned to stations on a purely random basis, it may lead to infeasible schedule because they break one or more precedence constraints It is the characteristic of the genetic algorithm that infeasible solutions are often generated by both crossover and mutation operators Anderson & Ferris have used a penalty function to drive the solutions toward feasibility They found this penalty approach to be most effective

4 2 3 Fitness and Selection for the mating pool

Fitness is a measure which determines how likely it is that an individual survives into next generation, or is selected for mating Anderson & Ferris defined the value of a solution as

$$v_i = \max_{i=1}^{N}(S_i) + kNv$$

where S_i = total time for tasks assigned to station i, *and* N = number of precedence violations, k = a constant set to largest task time and N = population size

The fitness of a solution is defined as $f_i = \exp(-hv_i)$, where h is chosen to make fitness lie in a particular range With the aim of producing a fitness distribution, linear scalıng ıs done Define the lınear scalıng by $F_i = m f_i + c$ ($i = 1$ to N) , havıng followıng propertıes

1 $\sum_{i=1}^{N} F_i = 1$

2 $\max_{i=1}^{N} F_i = \lambda \sum_{i=1}^{N} F_i / N$

3 $F_i = 0$ for all i

λ = scale factor to control speed of convergence

With $m = \dfrac{\lambda - 1}{N(\max_{i=1}^{N} f_i - \sum_{i=1}^{N} f_i)}$ $c = \dfrac{N(\max_{i=1}^{N} f_i) - \lambda \sum_{i=1}^{N} f_i)}{N(N(\max_{i=1}^{N} f_i) - \sum_{i=1}^{N} f_i)}$ and $\lambda = \min(\lambda_g, \lambda_k)$,

the three propertıes are satısfied where λ_g = the gıven value of scalıng,

and $\lambda_k = \min_k \left\{ \dfrac{((\max_{i=1}^{N} f_i) - f_k}{N(\max_{i=1}^{N} f_i - \sum_{i=1}^{N} f_i)} \middle| f_k \leq \sum_{i=1}^{N} f_i / N \right\}$

The method used to select ındıvıduals called stochastıc samplıng ıs descrıbed by Baker (1987) In thıs method, the member of the populatıon are randomly reordered Then, we assıgn to each ındıvıdual an ınterval proportıon to ıts fitness Paırs of ındıvıduals are removed from the matıng pool and recombıned usıng crossover and mutatıons

4 2 4 Cross Over And Mutatıon

Anderson & Ferrıs made lımıted experıments wıth dıfferent crossover mechanısm (Grefenstette, 1987), but dıd not found to gıve substantıal ımprovements over the more standard crossover mechanısm descrıbed ın the thırd chapter When a paır of ındıvıduals are selected for matıng cross over occurs at a sıngle random poınt wıth probabılıty p_c ,and with probability 1-p_c , the offsprings are identical to parent To allow mutation particular elements in the string are changed by plus or minus one with a small probability p_m If either of the offsprings have a worse fitness than that individual is not retained and, instead one of the parents is allowed to continue unchanged into the next generation Clearly by forcing the worst individual in each generation to be no worse than that of the previous generation will speed up the convergence of the algorithm If convergence is too fast we are likely to end up with a poor quality solution The rate of convergence, hence is controlled by scaling mechanism

4 2 5 Experimental Results of Anderson's GA

The initial population is generated by two ways In the 1st scheme the starting population was generated entirely on a random basis In the 2nd scheme, it generated a set of initial solution using a method due to Arcus (1966) They found the latter extremely effective In a significant proportion of cases, the initial set of Arcus solution contains at least one which is never improved upon by the GA, and in other cases the best of the Arcus' solutions is never far from the best solution found Thus, the GA implementation does not seem to be very effective Moreover, Anderson and Ferris have not spoken on the quality (optimality) of their solutions Hence, there is a scope of developing a genetic algorithm which can be more effective in terms of giving optimal or near-optimal results The present thesis work is a step in the same direction

4 3 GENETIC ALGORITHM DEVELOPED

Anderson and Ferris are the only researchers who have used GA to solve the assembly line balancing problem, however their results were inferior (or rather not superior) when compared to other better heuristics available till date The algorithm proposed here is found to be better than that of Anderson and Ferris and is expected to give quality solutions The detailed genetic algorithm follows

4 3 1 Data Storage

The ınput parameters for the SALBP-1 are C, the cycle-tıme n, number of tasks , $p_ı$, the processıng tımes of tasks , a number of precedence relatıons and paırs $(ı\text{-}j)$ ıllustratıng the actual precedence network The precedence matrıx ıs often very sparse, hence the network of the tasks ıs stored ın Forward and Reverse Star representatıon Thıs results ın mınımal memory utılızatıon Once a complete populatıon of new generatıon ıs ready, the parent populatıon (and hence the memory space occupıed by ıt) ıs destroyed In thıs way the memory space ıs used very effıcıently and optımally

4 3 2 Inıtıal Populatıon Generatıon

Three schemes for generatıng the ınıtıal populatıon have been developed and are descrıbed below

4 3 2 1 IPG-1 Scheme to generate Inıtıal Populatıon

(ı) Define *ındegree* of each node ın the network as the number of ıncomıng arcs to ıt Set $ı = 1$

(ıı) Search for all the nodes (tasks) ın the network whıch have zero ındegrees

(ııı) Store all such node numbers (task ıd) ın a set called *feasıble* or *elıgıble task set*

(ıv) Randomly select one task from the elıgıble task set

(v) Assıgn the task selected ın step (ıv) at the $ı$th place of a strıng of length n

(vı) Delete thıs task from the elıgıble task set Set ındegree of thıs task = -1

(vıı) Reduce the ındegrees of all the ımmedıate processors of thıs task by 1

(vııı) Increment $ı$ by 1

(ıx) Repeat steps (ıı) to (vııı) tıll the feasıble task set ıs not empty, or $ı \leq n$

The strıng thus generated ıs one member of the ınıtıal populatıon

(x) Repeat the steps (ı) to (ıx) N tımes to generate an ınıtıal populatıon of N strıngs

4 3 2 2 IPG-2 Scheme to generate Initial Population

(i) Define *indegree* of each node in the network as the number of incoming arcs to it Set $i = 1$ Initialize the remaining cycle time, *rem_time* = cycle time

(ii) Search for all the nodes (tasks) in the network which have zero indegrees

(iii) Store all such node numbers (task id) in a set called *feasible* or *eligible task set*

(iv) If there is no task in the feasible set having processing times less than rem_time, then, assign rem_time = cycle time

(v) Store all the tasks in the eligible set having processing times less than rem_time in a set called *greedy set*

(vi) Randomly select one task from the greedy set

(vii) Assign the task selected in step (vi) at the ith place of a string of length n

(viii) Delete this task from the eligible task set Set indegree of this task = -1

(ix) Delete all the tasks from the greedy set

(x) Increment i by 1

(xi) Repeat steps (ii) to (x) till the feasible task set is not empty, or $i \leq n$

The string thus generated is one member of the initial population

(xii) Repeat the steps (i) to (xi), N times to generate an initial population of N strings

4 3 2 3 IPG-3 Scheme to Generate Initial Population

This scheme generates a population of $10N$ using IPG-2, and selects the best 10% members as the starting initial population The factor of $k_{IPG\ 3}$, taken equal to 10 may be changed appropriately to alter the quality of the starting population For example, choosing 5% out of $20N$ (i e , $k_{IPG\ 3} = 20$) will yield a better initial population in terms of fitness values However, CPU time to generate the initial population will increase linearly

The average fitness value of an individual of scheme-2 is more than that of scheme -1 Whereas, the initial population of IPG-3 is superior than the remaining two in terms of both the average and the best fitness value The fitness of the members of initial population are evaluated in the following way

4 3 3 Fitness Measure

Fitness value of an individual string (member) is dependent only on one parameter, i e , the number of stations used Say, a member from the initial population (as generated by schemes discussed above) has genetic encoding 1 3 2 5 6 4 8 7 9 This implies that the tasks are to be assigned to station no 1 without violating the genetic order of the chromosome (individual) Once no more tasks can be assigned to station no 1 and there remain many unassigned task, switch to station no 2 Proceed similarly till all the tasks are allocated to a station or the other Then, the fitness value of this individual is defined by the following evaluation function

$$f_i = (S_{max} - S_i + 1)^k \quad , i = 1 \quad N$$

where S_i = no of stations used by the ith member of the population

S_{max} = no of stations used by the worst member of the population

k = a regulating parameter whose value is to be selected properly after experimental trials

If $k = 0$, fitness of all the members are equal

If $k = 1$, fitness of a member increases linearly as the no of station used decreases

If $3 \geq k \geq 1$, fitness of good members increases exponentially

If $k \geq 4$, results in a bias and therefore is not practicable

4 3 4 Parent Selection Scheme

The mating pool consists of the parent population members The parents are selected from the mating pool using roulette wheel selection procedure This method allows a relatively greater chance of mating of members having higher fitness measure The roulette wheel parent selection method has been described in detail in the third chapter The definition of the fitness function given in Sec 4 3 2 gives an extra power to the use of roulette wheel selection scheme k the regulatory parameter if kept equal to zero, changes roulette wheel selection scheme to a purely random parent selection scheme With a higher value of k (for example 2) roulette wheel scheme chooses good parents with a higher probability

4 3 5 Crossover schemes

Two major types of cross over schemes have been proposed here CROSS-1 and CROSS-2 CROSS-2 is purely original greedy crossover scheme — greedy in the sense that it drives the children to a better fitness value Whereas CROSS-1 employs few indexing criteria suggested by various researchers who have attempted on the assembly line balancing problem It, also is a greedy crossover scheme and incorporates several new concepts viz p_{i0} biasing etc Eight such criterion are employed, hence 8 versions of CROSS-2 are produced

4 3 5 1 CROSS - 1

(i) Say the members selected by roulette wheel parent selection scheme (for the network in Fig 2 1) are 1 3 5 9 2 5 6 8 7 10 and 1 4 2 6 3 5 8 7 9 10

(ii) One point crossover operator has been employed here For detailed description see Chap 3 The position of the crossover point can be selected in two ways

- [A] The crossover point is a random number selected between 1 and n
- [B] The crossover point is biased to be near the start of the string Or in other words, it is a random number between 1 and 0 5n Refer to Point (4) of Section 4 3 5 2 also

(iii) Let the crossover point be 4 Assign i = 4 (i e the crossing point position) Two children are produced from the crossover operation These resultant children or sequences of tasks may violate the precedence constraints because of the crossover operator Hence these child are not feasible, hence we will temporarily call them as *foetus* The foetus produced are as follows -

Parent 1 1 3 4 9 |2 5 6 8 7 10 Foetus 1 1 3 4 9 | 3 5 8 7 9 10

$\Rightarrow$

Parent 2 1 4 2 6 |3 5 8 7 9 10 Foetus 1 1 4 2 6 | 2 5 6 8 7 10

(iv) The foetus produced are not feasible There are three reasons for the infeasibility

(a) The precedence constraints are violated as we move across the crossover point

(b) Repetition of tasks occur (e g , task no 3 is positioned at 2nd and 5th places of Foetus-1

(c) Few tasks are completely missing from the foetus string (e g , task no 2 and 3 are missing from Foetus-1)

The deficiencies (b) and (c) of the foetus is removed, and hence the foetus develops into an *embryo* A method of doing this is described in Point(2) of Sec 4 3 5 2 Whereas, deficiency (a) remains in the embryos The steps (v) to (xv) are intended to remove this deficiency thus developing the embryo into a child The embryos thus emerged using Point(2) of 4 3 5 2 are -

Embryo 1 1 3 4 9 |2 5 8 7 6 10 Embryo 2 1 4 2 6 | 3 5 9 8 7 10

(v) Consider embryo 1 for the purpose of description The order of appearance of tasks on the left side does not violate the precedence constraints because the starting parent string were feasible Precedence relations are violated as soon as we move from left side of the crossover point to the right side However, the genetic material in the embryo falling on the right side of the crossing point destroys the feasibility Hence the genetic material on the right hand side of the crossover point has to be reorganised

(vi) Transfer all the genetic material of Embryo-1 (which is on the left side of the crossing point) to Array-A Hence, at present (i e , for the first iteration Array-1 = {3, 5, 8, 7, 9, 10}

(vii) Child-1 is an empty string Transfer all the genetic material of embryo-1 (which is on the left side of the crossing point) to Child-1 Now Child-1 is 1 3 4 9 $| \rightarrow$_ _ _ _ _ _

Only i = 4 tasks appear in the child string Mark all those tasks in the precedence network A marked task implies that it has been assigned to a station or the other Say, 2 stations have been used to accommodate all these marked tasks Calculate the remaining cycle time, *rem_time* of the current station Say rem_time = 18

(viii) Increment i by 1

(ix) Store all the unmarked tasks which are eligible (i e, which do not violate the precedence constraints) to appear at the ith position of the child string, in Stack-A (In the first iteration, Stack-A = {2, 6})

(x) If there is no task in Stack-A, whose processing time is less than rem_time, then, rem_time = cycle time That is, move to a new station

(xi) Copy all those tasks in Stack-A having processing times less than rem_time in Stack_B (In the first iteration, Stack-B = {2, 6})

(xii) Assign a *numerical score* $n(x)$ to all the unmarked tasks in the network

Remark A particular type of numerical score is selected, e g, positional weight scheme These numerical scores or indexing criterion or weightages for tasks have been suggested by various researchers in the past, and are given in Table 4 1

(xiii) Task-A is the task from Stack-A having the highest numerical score If there are more than one such tasks, then one is selected randomly out of those Call it Task-A

(xiv) Traverse in Array-A (from the 1st position of the array) to locate the first eligible task that can be assigned as the ith element of the child string A task can be said eligible, if it lies in Stack-A Call the task thus located as Task-B

(xv) Generate a real number a $0 \le a \le 1$

If $a > p$ Select Task-A

If $a \le p$ Select Task-B

(xvi) Update the precedence network by marking the task selected in step(xv)

No	Name	Description
1	Positional weight (Helgeson & Bernie)	The sum of the tasks for x and all tasks that must follow it
2	Reverse positional weight (Hackman Wee & Magazine)	The sum of the task times for x and tasks which precede it
3	Number of followers (Tonge, 1965)	The number of tasks that follow task x
4	Number of immediate followers (Mastor Tonge)	The number of tasks that immediately follow task x
5	Number of predecessors (Hackman, Wee and Magazine)	The number of tasks that precede task x
6	Work element time (Arcus)	The processing time of task x
7	Random basis (proposed)	The numerical score = 0 for all tasks

Table 4 1 Numerical Score functions $n(x)$

(xvii) Delete the task selected in step (xv) from Array-A Clear Stack-A and Stack-B

(xviii) Repeat steps (viii) to (xvii) till all the vacant positions of the child string is filled That is, continue as long as $i \leq n$

(xix) Repeat steps (v) to (xviii) to produce Child-2 For this, take Embryo-2 as the starting string instead of Embryo-1

Thus two child are produced from two parents with this crossover scheme

4 3 5 2 Additional features of CROSS-1

(1) CROSS-1 is a greedy crossover scheme It is greedy in the sense that it tends to produce children having better (if not worse than parents) fitness values

(2) Step (iv) of CROSS-1 mentions a procedure of developing the foetus to an embryo The mechanism is as follows -

Parent 1	1 3 4 9 | 2 5 6 8 7 10		Foetus 1 1 3 4 9 | 3 5 8 7 9 10
		$\Rightarrow$	
Parent 2	1 4 2 6 | 3 5 8 7 9 10		Foetus 1 1 4 2 6 | 2 5 6 8 7 10

The detailed steps are -

- Transfer the left portion of Foetus-1 in Array-1 Hence, Array-1 = {**1**, 3, **4**, 9}
- Transfer the left portion of Foetus-2 in Array-2 Hence, Array-2 = {**1**, **4**, 2, 6}
- Highlight the elements that are common to both the arrays
- Modify the arrays by deleting the highlighted (bold faced) elements
- Thus, Array-1 = {3 9} and Array-2 = {2, 6}
- Maintain a 1-to-1 correspondence between the elements of Array-1 and the elements of Array-2 That is, the *i*th element of Array-1 corresponds to the *i*th element of the Array-2

$$\begin{array}{lcc} \text{Array-1} & 3 & 9 \\ & | & | \\ \text{Array-1} & 2 & 6 \end{array}$$

- Come back to Foetus-1 1 3 4 9 $\underline{3}$ 5 8 7 $\underline{9}$ 10

  If there is any element (on the right hand side portion) of the foetus string common to Array-1, underline it Repeat this logic for Foetus-2 Hence, Foetus-2 is 1 4 2 6 $\underline{2}$ 5 $\underline{6}$ 8 7 10

  Interchange the underlined elements of Foetus-1 with those of Foetus-2 following 1-to-1 correspondence Do vice-versa Thus foetus emerges into an embryo

(3) The steps (v) to (xvii) describe a procedure to generate Child-1 (and Child-2 similarly) from the infeasible embryos If carefully analysed, one will notice that only the RHS (right hand side) portion of the embryo is modified (or altered) to form a child And, the process of assigning tasks is from left to right This is called *Forward balancing* However, the LHS part of the embryo remains untouched (hence, unimproved by the crossover)

Backward balancing is done to provide improvement on the LHS part of the string Thus, we may obtain two strings by embryo (say embryo-1) one by forward balancing and one by backward balancing The string better (which uses a lower number of stations) out of the two is the child (say Child-1) The other string is destroyed Similar procedure may be used to produce Child-2

<u>Backward balancing</u>

A copy of precedence network is made The direction of all the arcs in this network are reversed If crossing point is, say $i = 4$, then, first 4 elements of the embryo from the left are fixed That is, the partially formed embryo is _ _ _ _ _ _ ← | 8 7 9 10 The tasks are assigned in the direction of the arrow using the reversed network The logic of assigning tasks is similar to the steps (v)–(xvii) of CROSS-1

(4) Step (ii) of CROSS-1 speaks on the selection of the position of the crossover point Point[B] of Section 4 3 4 1 says that crossover point should lie in the 1st half of the string What is the logic behind it? For example, the crossing point is 7, and the length of the string is 10 When carefully analysed, one will notice that the improvement (or change) will occur only on the 30% (portion on the RHS of the crossover point) portion of the child string The 70% (on the LHS of the child string) portion remains untouched and unimproved

<u>*Biasing*</u> Hence, we may wish to bias the crossing point to be placed near to the starting end of the string Apart from Point[B] of step (ii), another method has been

devised to force the crossing point take a place near the starting end of the string The method can be described by the following probability function -

$$P_{cpp}(i) = \frac{(n-i+1)^q}{\sum\limits_{i=1}^{n}(n-i+1)^q}$$

This function ensures that $P_{cpp}(1) \geq P_{cpp}(2) \geq \quad \geq P_{cpp}(n)$ This implies that probability of crossing point lying near to the start of the string is higher

$P_{cpp}(1)$ = The probability that i is the position selected for crossover

q = A parameter to control the degree of biasing the crossing point

If $q = 0$ then $P_{cpp}(i) = P_{cpp}(j+1) \quad \forall\, i$

If $q > 0$, then $P_{cpp}(i) > P_{cpp}(i+1) \quad \forall\, i$

That is if q is very high, the crossover point will definitely be 1, i e totally biased

(5) Step (xii) of CROSS-1 demands selection of a particular type of numerical score Seven such scores have been given in Table 4 1 When a particular criteria say positional weight is selected, the same criteria is used throughout for producing all children and in all generations Thus 7 criteria gives 7 versions of CROSS-1, i e, CROSS-1 1 to CROSS-1 7

If, for each crossover operation we select a new type of indexing criterion (randomly selected from those 7), it will result into another version of CROSS-1 Let it be named CROSS-1 8 That is, while producing a new child every time the type of numerical score is selected randomly from the set of 7

(6) The step (xii) of CROSS-1 assigns a numerical score to all the potential and eligible tasks The task with the highest score is selected Each time we have to select a task (to place at the next position in the child string), the numerical scores are updated The method, thus used in CROSS-1, is called IUFF, *Immediate Update First Fit* There are two more methods for assigning numerical scores -

- GFF, or *Generalized First Fit* In this scheme the numerical scores are assigned and are not updated till we move to the next station The scores of all the unmarked tasks are updated by step (xii) only when we switch to the next station
- R & A, *Rank and Assign* In this scheme, the numerical scores are assigned to all the unmarked tasks and are never updated by step (xii) in future

(7) It is possible that good parents may produce bad child If one child is worse than both the parents replace that child by the better parent (amongst both) However , if both the child are worse than the worst parent, replace both the children by the parents

(8) Features (2) to (7) are expected to improve the results (in terms of fitness value) of CROSS-1 However, it is at the discretion of the user whether or not to include these features in the cross over scheme

4 3 3 3 CROSS-2

Steps (i) to (iii) of the crossover scheme, CROSS-2 are same as that of CROSS-1, hence repetition of those steps is not being done here The further steps are as follows

(iv) The foetus produced are not feasible The reasons for infeasibility are described in the points (a) (b), (c) of step (iv) of CROSS-1

(v) Consider Embryo-1 for the purpose of description The order of appearance of tasks on the left side does not violate the precedence constraints because the starting parent string were feasible Precedence relations are violated as soon as we move from left side of the crossover point to the right side However, the genetic material in the embryo falling on the right side of the crossing point destroys the feasibility Hence the genetic material on the right hand side of the crossover point has to be reorganised

(vi) Find s the number of stations that have been used to accommodate all the tasks on the LHS of the crossover point (in Foetus-1) Find n_s the number of tasks in the station no 1 For the first iteration in the case of Foetus-1, $s = 2$ and n_s=1

(vii) Modify the value of i as $i = i - 1$ Initialize $j = 1$

(viii) Child-1 is an empty string Transfer the first i tasks of Foetus-1 to Child-1 Hence the incompletely formed child, Child-1 is 1 3 4 9 |→_ _ _ _ _ _ This is called as State-A of Child-1

(ix) Mark all the tasks of Child-1 (State-A) in the precedence network The marked network is called as State-1 of the precedence network

(x) Since we are at the start of station assign a value to the remaining cycle time as rem_time = cycle time

(xi) Store all the unmarked tasks eligible (which do not violate the precedence constraints) to appear at the ith position of the child string in Stack-A In the first iteration Stack-A = {2, 6}

(xii) Copy all those tasks in Stack-A having processing times less than rem_time in Stack_B In the first iteration, Stack-B = {2, 6}

(xiii) Randomly select a task from Stack-B, and assign it to the ith place of the child string

(xiv) Subtract the processing time of the task selected in step (xiii) from rem_time Update the network by marking this task in the precedence network

(xv) Increment i by 1

(xvi) Repeat steps (x) to (xv) till Stack-B is not empty If Stack-B is empty move to the next step

(xvii) The collection of tasks selected in steps (x) to (xvi) is store in Task-Set$_{j,ns}$ There are many sets of task assignments possible for a station Task-Set$_{j,ns}$ is one such assignment (out of possible many) of tasks for station no n_s

(xviii) The present status of the precedence network (in terms of the tasks marked or unmarked) is called as State-2 of the network

(xix) If $j < n_{trial\_sets}$ update the network to its State-1

Remark n_{trial_sets} is the number of trial combinations (of tasks) made for a station before the final assignment approved (Say $n_{trial\_sets} = 10$)

(xx) If $j < n_{trial\_sets}$ increment j by 1 Repeat steps (ix) to (xviii)

If $j \geq n_{trial\_sets}$ go to step (xxi)

(xxi) Evaluate all the n_{trial_sets} (i e 10) number of possible task sets The best task set is the one having the highest work content (i e the sum total of the processing times of all the tasks in that set) This is the best possible combination of tasks for station no n_S Say n_{tss} be the number of tasks in this set selected Fill the places (i+1) to (i+n_{tss}) of the child string by the elements of the best task set

(xxii) Update the precedence network to its State-2

(xxiii) If Stack-A is not empty $\rightarrow$ Increment n_S by 1, i e , move to the next station for further task assignments

(xxiv) Repeat the steps (x) to (xxiii) till Stack-A is not empty i e , repeat till no more tasks remain as unassigned

(xxv) The steps (v) to (xxiv) result in the complete formation of the child string, Child-1 The same steps may be repeated to get Child-2 from Foetus-2

4 3 5 4 Additional features of CROSS-2

All the additional features (except Point nos 2, 5 &6) of CROSS-1 described in Sec 4 3 4 2 are also true for CROSS-2

4 3 6 The Mutation Operator

The mutation scheme developed here employs swapping of tasks in the string in a predetermined manner The details are as follows

(i) Say the child string to undergo mutation has the following genetic structure -

Child string 1 4 2 | 5 6 3 8 | 7 9 10

Station-1 Station-2 Station-3

The symbol '|' represents a demarcation line between two stations

(ii) Replace all such '|' with '↑' the tasks lying on the sides of a '↑' when interchanged do not violate precedence constraint Thus the intermediate string is -

Intermediate swap

Child string 1 4 2 | 5 6 3 8 ↑ 7 9 10

Station-1 Station-2 Station-3

(iii) Swap the tasks lying on the sides of '|', with each other Then the string after the mutation has occurred is as follows -

Child string 1 4 2 | 5 6 3 7 | 8 9 10

<u>Remark</u> We could also have used the random swapping of tasks, i e , swap any randomly selected task with any other randomly selected task We could have done this, no doubt, but there are high chances that the string after mutation will be infeasible (due to the violation of precedence constraints) However the swapping scheme suggested in step has following positive features

- Swapping of tasks within a station has no advantage because swapping such tasks will not result in a different station assignments For example, consider station no 2 of the child string (prior to mutation) If the tasks placed at the 4th and 5th positions of the sting are swapped, the task set for station 2 is still {2, 6, 9}, i e , it is unchanged Hence, no change is implemented by such swapping Thus swapping between the tasks of different stations is justified
- The tasks which are immediate neighbours hardly destroy the feasibility of the string when interchanged or the chance of doing so is very low The reason is that they have a high probability to appear in Stack-A (it contains tasks

eligible to appear at the next position of the child string Refer to step(ix) of CROSS-1) simultaneously The tasks of Stack-A do not have any precedence relationship amongst each other

As the proximity of any two tasks in the string increases the chance that they will not hamper the feasibility of the string when interchanged (or swapped) increases For example, the first task (task no 1) and the last task (task no 10) are bound to produce infeasible string when swapped with each other Thus swapping amongst the adjacent tasks is also justified

- Even then there is a good chance that precedence constraints may get violated Hence the step (ii) recommends that only the tasks lying on the sides of '↑' are selected for being mutually swapped Swapped is not done for '|' where precedence constraints are known to be violated

(iv) Reassign the places of '|', in the mutated string (as per the values of the cycle time and task processing times) in reference to the restructuring of the genetic material caused by the mutation

4 4 A NOTE ON THE GENETIC ALGORITHM DEVELOPED

The model of the genetic algorithm proposed above can be summarised by the following pidgin algorithm

```
Generate the initial population ( use IPG-1, 2  3)
    Initialize the parent population as the initial population
    Repeat
        for each individual $i$ do
            Evaluate $f(i)$, the fitness measure
        Repeat
            Select individual $j$ and $k$ for mating
            ( Roulette Wheel Parent selection scheme )
            do crossover and mutation operations ( CROSS-1  2 etc )
            Move both the child produced to the new generation population
        Until number of members in new generation = $N$
        Replace the individuals of parent population by child population
        New generation population set is now empty
    Until population variance of the new generation is small
    Select the best member of the final generation as the solution of GA
```

The crossover schemes, CROSS-1 & 2 are improvization based crossover schemes These schemes are being called greedy for the very reason, that, the idea of improvement or optimization has been fused within the crossover operator This idea of optimization has been incorporated to a greater extent in CROSS-2 in which the best likely task assignments for a station is found before moving to the next station

Since the crossover schemes developed are greedy by nature, it is suggested to adopt a value near to 1 for p_{cr}, the crossover probability The mutation operator suggested above is devised in such a way such that the resultant string is not infeasible

However the mutation operator hardly offers any improvement Moreover it reduces the fitness measure of the members (which has been increased by the crossover operator) in most of the cases Hence a low probability of mutation p_m of near to zero is chosen

However to allow diversity in the genes of the population, *immigration* of members from a *foreign population* can be done This will likely improve the performance of EGAALB

Immigration The *immigrant* to the child population is the best member of the *foreign population* After every reproduction process, a random number is generated, and if it is less than p_i, (the *probability of immigration*), then immigration is allowed, otherwise not Whenever immigration has to occur a foreign population of k_i, (number of members in the foreign population) members is generated using any of the 3 initial population generation schemes The member having the best fitness measure, out of these k_i is allowed to immigrate That is the immigrant replaces the worst member of the child population It is expected that, if incorporated in EGAALB, this feature will improve the results

A number of control parameters have been provided in the genetic algorithm developed which offers a great deal of control over the working of the GA The parameters are

(1) N, the population size, (2) n_{gen}, the number of generations, (3) k, to control the fitness function and intensify (alter) the power of roulette wheel parent selection scheme (4) q to control the degree of biasing the position of crossover point position, (5) $n(x)$, the numerical score function, (6) n_{trial_set}, to control the optimal seeking power of the crossover scheme, CROSS-2, (7) *elitism*, i e , replacing the worst member of the child population with the best member of the parent population, (8) replacing bad children by good parents, (9) selection of the initial population generation scheme, (10) p_{ro}, the prob

Chapter 5

EXPERIMENTATION AND RESULTS

5 1 THE PROBLEM SET

To probe the capabılıty and the effectıveness of EGAALB, test problems were drawn from the standard set of 64 lıterature problems, assembled by Talbot & Patterson (1984) and used by Johnson (1988) Hoffmann (1992) and several others However, out of these 64 only 49 were attempted The remaınıng 15 problems due to Dar-El (11 task problems) Heskıa (28 task problems) and Kılbrıdge & Wester 45 task problems) could not be attempted because the problem data could not be accessed It ıs judged that thıs algorıthm wıll have an optımal solutıon for all these 15 problems sınce they are relatıvely easy

Whereas the 49 problems whıch have been selected, consısts of all those 'tough' problems whıch have been found dıfficult by several researchers The two most crıtıcal problems from the set of 64 problems are (ı) Tonge's problem of 70 tasks, cycle tıme = 176, and (ıı) Arcus' problem of 111 tasks, cycle tıme = 5755

5 2 THE EXPERIMENTAL PLAN

The genetıc algorıthm, whıch has been developed, has several control parameters and optıons These parameters are to be set properly for the desıred effıcıent workıng of the algorıthm ın terms of (a) qualıty of the solutıon, and (b) computatıon tıme The parameters selected after doıng excessıve experımentatıon are -

(ı) *N*, the populatıon sıze Several values of *N* have been trıed A small populatıon sıze doesn't gıve much dıversıty, thus curbıng the effıcıency of genetıc algorıthm and hence lowering the quality of solution Whereas a large value of N greater than 100 adversely increases the computation time and memory required A value of $N = 50$ has been found to be suitable

(ii) n_{gen} number of generations To avoid premature convergence of the population the number of generations is selected as $n_{gen} = 20$ However even lower values have also worked good The best member of the 20th generation population is taken as the final solution for the problem

(iii) k, the parameter which controls the fitness function The value of $k = 1$ is found to be best However $k = 2$ also gives acceptable results for most of the problems

(iv) q the parameter to bias the position of the crossover point near the start of the string It is found that biasing of the crossover point results in an inferior solution However, when crossover point is randomly placed at any position (between 1 and n) of the string the individuals produced are of higher quality Thus $q = 0$

(v) n_{trial_set} number of trial sets taken for a station before the final assignment A value of $n_{trial\_set} = 10$ is found to be best

(vi) *Elitism* This option is chosen i e , the worst member of the child population is replaced by the best member of the parent population

(vii) The option that bad child are replaced by good parents during reproduction (crossover) is not chosen Replacing the child by parents result in an early pre-mature convergence On the other hand when children are not replaced by parents, a good amount of diversity is available in the population which is needed Moreover, the option other than the one selected failed to give optimal balances in 2 cases

(viii) p_m, the probability of mutation It was observed that the mutation operator worsens the quality of the individuals The reason is that the improvement done by the greedy crossover operators is neutralized by the mutation operator (which is not a optimization based operator) Thus $p_m = 0$ serves the purpose best

(ix) p_{cr}, the crossover probability Since the crossover schemes are designed to provide improvement to the parent solution strings the value of p_{cr} should be no less than one i e $p_{cr} = 1$

5 3 COMPUTATIONAL INVESTIGATIONS

A wide experimentation has been done in order to achieve the best setting of the parameters Here the performance of EGAALB will be presented with the above parameter settings and options, when different crossover schemes and initial population generation schemes are considered

First of all, an analysis of the initial population generated by the 3 schemes will be beneficial before moving ahead Table 5 1 presents an analysis of the initial population distribution in terms of the quality or fitness measure It tells whether the optimal solution to the problem is contained in the initial population or not Column (1) shows the number of stations used by the best member of the initial population A mark (#), if present, depicts that the best member found is sub-optimal Column(2) gives the average number of stations used by a member It is easy to conclude from Table 5 1, that the initial population generated by the 3 schemes are in the following order IPG-1 < IPG-2< IPG-3 Or, the initial population generated by the third scheme contains optimal members most of the time

However, there is one problem with IPG-3 This scheme generates members which are often identical to (or copy of) each other On the other hand, IPG-1 is inferior in terms of fitness measure but offers a good amount of chromosomal diversity in its population Diversity amongst the individuals is very important for the efficiency of an improvement based genetic algorithm Then, the question arises Is it wise to choose IPG-1 ? The answer is yes The idea is that the crossover schemes will continuously improving the species over generations resulting in highly evolved individuals

Researcher's Name	No of task	cycle time	Initial Population Generation Scheme					
			IPG-1		IPG-2		IPG-3	
			(1)	(2)	(1)	(2)	(1)	(2)
Merten	7	6	6	6	6	6	6	6
		7	5	5 7	5	5 44	5	5
		8	5	5 7	5	5 44	5	5
		10	3	3 74	3	3 64	3	3
		15	2	2 3	2	2 24	2	2
		18	2	2	2	2	2	2
Bowman	9	20	5	5	5	5	5	5
Jaeschke	9	6	8	8 38	8	8	8	8
		7	7	7 58	7	7 14	7	7
		8	5	6 20	5	5 76	5	5
		10	5	5	5	5	5	5
		18	3	3	3	3	3	3
		20	2	2 38	2	2 18	2	2
Jackson	11	7	8	8 70	8	8 40	8	8
		9	6	6 26	6	6 26	6	6
		10	6 #	6	5	5 82	5	5
		13	4	4 42	4	4 24	4	4
		21	3	3	3	3	3	3
Mitchell	21	14	9 #	10 3	9 #	9 80	9 #	9
		21	5	6 08	6 #	6	5	5 74
		35	4 #	4	4 #	4	3	3 86
Sawyer	30	25	15 #	16 82	14	15 56	14	14 74
		27	14 #	15 46	13	14 38	13	13 96

		30	13 #	13 62	12	12 60	12	12
		36	11 #	11 52	10	10 74	10	10
		41	9 #	10 02	9 #	9 2	9 #	9
		54	7	7 04	7	7	7	7
		75	5	5	5	5	5	5
Tonge	70	176	24 #	25 6	22 #	23 14	22 #	22 36
		364	10	11 08	10	10 44	10	10
		410	9	9 92	9	9 04	9	9
		468	8	8 70	8	8 02	8	8
		527	7	7 88	7	7 12	7	7
Arcus	83	5048	17 #	17 22	16	16 70	16	16
		5853	14	14 68	14	14 02	14	14
		6842	12	12 92	12	12 78	12	12
		7571	11	11 02	11	11	11	11
		8412	10	10	10	10	10	10
		8998	9	9 08	9	9	9	9
		10816	8	8	8	8	8	8
Arcus	111	5755	31 #	33 52	28 #	29 38	28 #	28 66
		8847	19 #	20 38	18	18 90	18	18
		10027	17 #	17 92	16	16 76	16	16
		10743	15	16 74	15	15 66	15	15
		11378	14	15 46	14	14 42	14	14
		17067	10 #	10	9	9 68	9	9

Note (1) Average number of statıons used by a member of the ınıtıal populatıon

(2) No of extra statıons requıred by the best member of the ınıtıal populatıon over the optımal number of statıons requıred

(#) The best member of the ınıtıal populatıon ıs sub optımal

Table 5 1 Analysıs of the Inıtıal populatıon

Table 5 2 shows the number of stations used by the best member of the final population (20th generation) for different crossover schemes (CROSS-1 1 to 1 8 & CROSS-2) Scheme IPG-1 is used for all the computations in the Table 5 2 It was found that the results with CROSS-2 are the best and it gives optimal answers for all the literature problems attempted

Table 5 3 presents the CPU times in seconds for different problems solved in seconds for various problems tried The experimentation was carried out on HP-9000 mini computer series in UNIX multi task sharing environment The CPU times presented in the table is exclusive of the input and output times It is conclusive from Table 5 3 that, the computation times for CROSS-2 are relatively more than CROSS-1 Total CPU time by EGAALB (using IPG-1 & CROSS-2) for all the 49 problems was less than two seconds (i e , 1 76s)

Tonge's problem of 111 tasks (cycle time = 5755s) is chosen to demonstrate how the genetic algorithm plays a role in the evolution of species Fig 5 1 illustrates how the average number of stations used by an individual decreases over generations

Fig 5 1 also shows how the number of stations used by the best member of a population decreases It can be seen that, the best member of generation no 1 uses 27 stations There are 3 such members This value of 27 stations, which is optimal, is never improved in further generations What happens is that the number of optimal members increase in further generations In fact, the optimal balance in the very first generation demonstrates the optimization power of CROSS-2 The CROSS-1 schemes are also greedy but their optimizing power is far low when compared to CROSS-2

Researcher's Name	no of tasks	cycle time	No of stations used with crossover shemes								
			1.1	1.2	1.3	1.4	1.5	1.6	1.7	1.8	2
Merten	7	6	6	6	6	6	6	6	6	6	6
		7	5	5	5	5	5	5	5	5	5
		8	5	5	5	5	5	5	5	5	5
		10	3	3	3	3	3	3	3	3	3
		15	2	2	2	2	2	2	2	2	2
		18	2	2	2	2	2	2	2	2	2
Bowman	9	20	5	5	5	5	5	5	5	5	5
Jaeschke	9	6	8	8	8	8	8	8	8	8	8
		7	7	7	7	7	7	7	7	7	7
		8	5	5	5	5	5	5	5	5	5
		10	5	5	5	5	5	5	5	5	5
		18	3	3	3	3	3	3	3	3	3
		20	2	2	2	2	2	2	2	2	2
Jackson	11	7	8	8	8	8	8	8	8	8	8
		9	6	6	6	6	6	6	6	6	6
		10	6*	6*	6*	6*	6*	6*	6*	6*	5
		13	4	4	4	4	4	4	4	4	4
		21	3	3	3	3	3	3	3	3	3
Mitchell	21	14	9*	9*	9*	8	9*	9*	9*	9*	8
		21	5	5	5	5	5	5	5	5	5
		35	3	3	3	3	3	3	3	3	3
Sawyer	30	25	14	14	14	14	14	14	15*	14	14
		27	13	13	14*	14*	14*	13	14*	13	13
		30	12	12	12	12	12	12	12	12	12

		36	10	10	10	10	10	10	10	10	10
		41	9*	9*	9*	9*	9*	9*	9*	9*	8
		54	7	7	7	7	7	7	7	7	7
		75	5	5	5	5	5	5	5	5	5
Tonge	70	176	23#	22*	23#	23#	23#	22*	23#	23#	21
		364	10	10	10	10	10	10	10	10	10
		410	9	9	9	9	9	9	9	9	9
		468	8	8	8	8	8	8	8	8	8
		527	7	7	7	7	7	7	7	7	7
Arcus	83	5048	16	16	16	16	16	16	16	16	16
		5853	14	14	14	14	14	14	14	14	14
		6842	12	12	12	12	12	12	12	12	12
		7571	11	11	11	11	11	11	11	11	11
		8412	10	10	10	10	10	10	10	10	10
		8998	9	9	9	9	9	9	9	9	9
		10816	8	8	8	8	8	8	8	8	8
Arcus	111	5755	29#	29#	29#	30@	29#	29#	29#	29#	27
		8847	18	18	18	18	18	18	18	18	18
		10027	16	16	16	17*	17*	16	17*	16	16
		10743	15	15	15	15	15	15	15	15	15
		11378	14	14	14	14	14	14	14	14	14
		17067	9	9	9	9	9	9	9	9	9

Note (*) = one station used over optimal, (#) = two stations used over optimal
(@) = three stations used over optimal

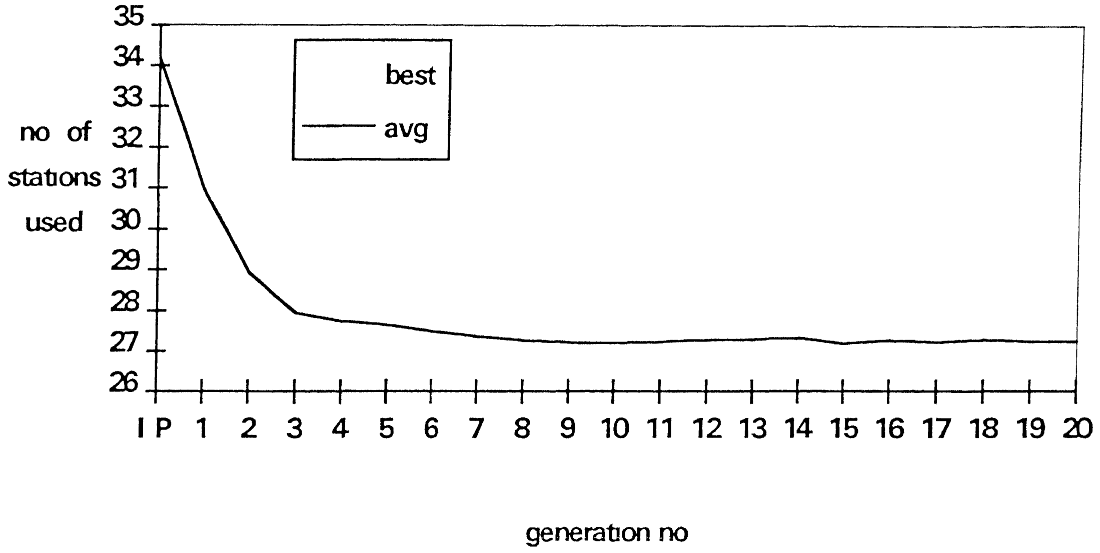
Table 5 2 No of stations used by EGAALB with diffrent crossover schemes

Researcher's Name	no of tasks	cycle time	CPU time required by crossover shemes								
			1 1	1 2	1 3	1 4	1 5	1 6	1 7	1 8	2
Merten	7	6	0 01	0 01	0 02	0 02	0 01	0 01	0 01	0 01	0 01
		7	0 02	0 03	0 01	0 01	0 01	0 01	0 01	0 01	0 0
		8	0 02	0 02	0 01	0 01	0 01	0 03	0 01	0 03	0 04
		10	0 03	0 02	0 01	0 02	0 02	0 01	0 01	0 00	0 02
		15	0 04	0 02	0 02	0 02	0 02	0 01	0 02	0 03	0 01
		18	0 01	0 01	0 01	0 00	0 00	0 02	0 02	0 03	0 04
Bowman	9	20	0 01	0 01	0 02	0 02	0 02	0 02	0 01	0 01	0 03
Jaeschke	9	6	0 03	0 01	0 01	0 01	0 02	0 03	0 01	0 02	0 03
		7	0 02	0 01	0 04	0 03	0 02	0 0	0 01	0 01	0 02
		8	0 03	0 02	0 03	0 02	0 01	0 03	0 04	0 01	0 01
		10	0 03	0 01	0 03	0 02	0 01	0 01	0 01	0 01	0 00
		18	0 02	0 02	0 02	0 03	0 01	0 01	0 01	0 01	0 03
		20	0 01	0 02	0 02	0 02	0 01	0 0	0 01	0 02	0 0
Jackson	11	7	0 01	0 01	0 03	0 03	0 03	0 01	0 02	0 02	0 01
		9	0 01	0 02	0 02	0 01	0 01	0 03	0 01	0 02	0 01
		10	0 02	0 03	0 02	0 02	0 01	0 01	0 01	0 01	0 03
		13	0 02	0 02	0 04	0 02	0 01	0 0	0 01	0 00	0 03
		21	0 02	0 03	0 02	0 04	0 01	0 01	0 02	0 01	0 04
Mitchell	21	14	0 01	0 03	0 03	0 02	0 02	0 02	0 03	0 02	0 04
		21	0 01	0 02	0 03	0 03	0 02	0 03	0 02	0 01	0 03
		35	0 03	0 03	0 01	0 01	0 02	0 0	0 01	0 00	0 03
Sawyer	30	25	0 03	0 02	0 01	0 01	0 01	0 01	0 01	0 02	0 03

		27	0 01	0 02	0 04	0 02	0 01	0 01	0 02	0 01	0 04
		30	0 01	0 01	0 01	0 02	0 01	0 01	0 03	0 02	0 0
		36	0 02	0 01	0 02	0 02	0 02	0 00	0 01	0 01	0 0
		41	0 01	0 01	0 01	0 03	0 02	0 03	0 03	0 01	0 03
		54	0 02	0 01	0 01	0 02	0 03	0 01	0 03	0 01	0 06
		75	0 02	0 02	0 01	0 01	0 01	0 0	0 02	0 01	0 0
Tonge	70	176	0 03	0 02	0 03	0 02	0 02	0 0	0 02	0 01	0 03
		364	0 03	0 04	0 01	0 03	0 00	0 01	0 01	0 03	0 05
		410	0 01	0 03	0 05	0 01	0 02	0 04	0 03	0 02	0 06
		468	0 00	0 00	0 04	0 01	0 02	0 02	0 01	0 01	0 03
		527	0 01	0 04	0 01	0 03	0 05	0 05	0 02	0 02	0 04
Arcus	83	5048	0 01	0 02	0 01	0 00	0 03	0 03	0 01	0 04	0 05
		5853	0 04	0 03	0 03	0 02	0 03	0 0	0 03	0 03	0 06
		6842	0 02	0 02	0 03	0 02	0 02	0 0	0 01	0 02	0 06
		7571	0 03	0 01	0 02	0 02	0 02	0 0	0 01	0 01	0 06
		8412	0 02	0 02	0 04	0 03	0 03	0 03	0 01	0 02	0 06
		8998	0 03	0 02	0 02	0 03	0 02	0 02	0 03	0 01	0 07
		10816	0 02	0 00	0 00	0 02	0 02	0 03	0 03	0 04	0 04
Arcus	111	5755	0 02	0 03	0 03	0 02	0 04	0 0	0 02	0 06	0 05
		8847	0 02	0 03	0 02	0 01	0 04	0 01	0 05	0 05	0 05
		10027	0 03	0 04	0 01	0 03	0 02	0 03	0 01	0 01	0 05
		10743	0 03	0 03	0 05	0 04	0 03	0 0	0 02	0 01	0 11
		11378	0 02	0 03	0 07	0 04	0 06	0 03	0 03	0 04	0 07
		17067	0 06	0 02	0 03	0 04	0 05	0 02	0 01	0 01	0 10

Table 5 3 CPU times in seconds for EGAALB with different crossover schemes

Fig 5 1 Vanation of the nurber of stations used over generations



Note best plot of the number of stations used by the best member of the ith generation
avg plot of the average number of stations used bv any member of the ith generation

Fig 5 2 Illustration of the variation of the population variance over generations

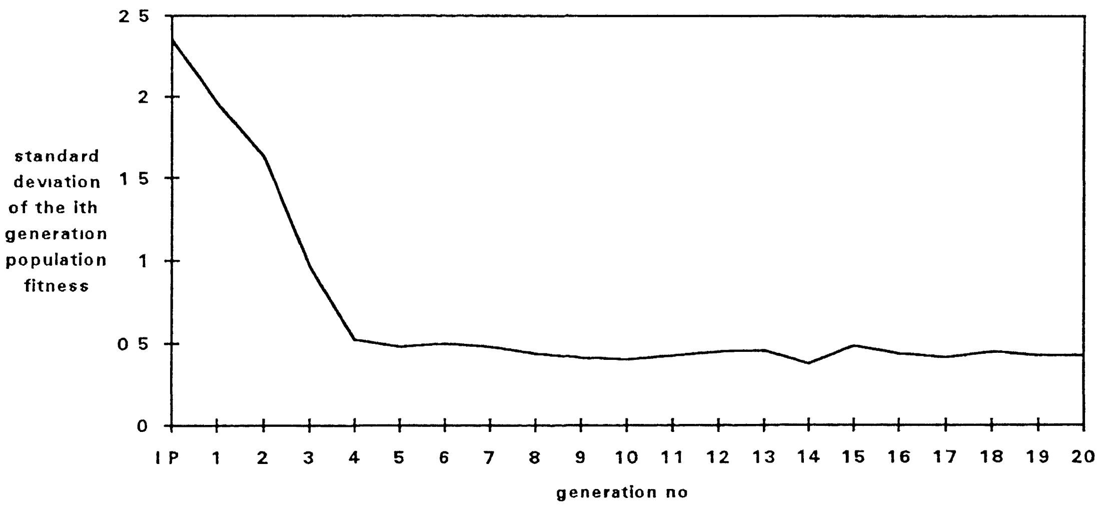


Fig 5 2 shows how the population variance or standard deviation decreases over generations The population has converged to 95% in the 30th generation However, our criteria is to stop after 20th generations and take its best member as the solution to the problem The best member of the 30th generation is also present in the 20th generation population

In 85% of the problems attempted, full convergence was achieved within five generations However the result has been demonstrated for the most critical case, i e , Arcus' 111 task problem (Fig 5 1 & 5 2)

The crossover schemes CROSS-1 1 to 1 8 fail to give optimal results for one-sixth of the problems when IPG-1 is used This combination of CROSS-1 with IPG-1 gave optimal answers for 92% of the problems In the remaining 8% of the problems (Mitchell 21/14, Sawyer 9/41, Tonge 70/176 and Arcus 111/5755), only one extra station was used over optimal solution Whereas, CROSS-1 schemes when combined with IPG-3 gave optimal solutions for 96% of the problems However the combination of IPG-1 and CROSS-2 provided optimal solution for all the problems

It was further noticed that the complexity or toughness of a problem is less dependent on the numoer of tasks, and is more dependent on the cycle time The problem becomes more difficult when the cycle time is near to the processing time of the largest task For example, crossover scheme CROSS-1 1 gives optimal results for all the 111 task Arcus problems except the one having cycle time = 176

5 4 WHICH IS THE BEST HEURISTIC IN THE LITERATURE ?

To determine the efficacy of the genetic algorithm, EGAALB, a need is felt to find out which is the best heuristic The idea is to compare the results of EGAALB with those of the best heuristic Baybars (1986), earlier, in his survey paper has attempted to determine which is the best heuristic for SALBP-1

Baybars (1986) classified the heuristics available in the literature in 4 categories (i) single pass decision rules (ii) composite decision rules (iii) backtracking decision rules, (iv) Optimal seeking decision rules The fourth category includes exact methods viz Integer Programming, Dynamic Programming etc which are non-polynomial time algorithms If a computational time constraint is implemented on these optimal guaranteeing algorithms, we can treat it as a heuristic variant of these optimization algorithm This is considered temporarily just for the sake of comparison with pure heuristic procedures therefore, such variant with time constraint of 3 0 s (AMDAHL 470/V8 CPU time, Baybars 1986) can be included in the fourth category as a type of heuristic He carried out extensive experimentation on 26 heuristic rules and the results of his investigations are illustrated in Figure 5 3 and Table 5 4

Referring to Table 5 4 , Baybars comments that Hoffmann heuristics give the best results for the main experimental problems When considering literature problems, MAG-1 was the best amongst the optimal optimal seeking exact decision rules It gives minimum stations when compared to all other heuristics and reported optimal balances for 55 problems It could not provide an optimal balance for 9 other within the time limit Amongst the pure heuristic procedures, MALB and COMPOSITE -13 were the best However, no heuristic provided optimal balances for all the literature problems MAG-1 and Hoffmann heuristics are reported to perform best on difficult data set 1 and set 2

In his paper, Baybars (1986) finally concludes that the variants of Hoffmann's (1963) precedence matrix approach followed by Dar-El's MALB is best amongst all the pure heuristic methods Of the optimal seeking methods, Magazine and Wee's branch and bound procedure using the heuristic IUFFD, and Talbot and Patterson's (1984) implicit enumeration procedure initialized with Hoffmann's procedures yields the best overall results

Hackman, Magazine & Wee (1988) present a heuristic which is efficient in terms of CPU time but not in terms of the quality of the solution However, Johnson (1980) presents an optimal seeking branch and bound exact algorithm called FABLE He also proposed a heuristic variant of the same which gave optimal answer for all problems less than 100 tasks but failed for a problem over 100 tasks

Hoffmann (1992) proposes a branch and bound algorithm, which in many cases is faster than the well known Hoffmann(1963) heuristic to which it is related Using implicit complete enumeration, ICE procedure with a 3 sec time window, Hoffmann obtained optimal solutions for all the literature problems except one (Arcus' 111 task problem with cycle time = 5755 s)

Anderson (1994) proposed a genetic algorithm for the assembly line balancing problem, but his purpose was just to demonstrate that GA can be used for such NP Hard Problems However optimality was not his prime objective

Therefore, it is easy to conclude that among the heuristic procedures, the one by Hoffmann(1963, 1992) are the best in term of quality of the solution

5 5 COMPARISON OF THE RESULTS

It is clear from the discussion of the previous article, that no heuristic has reported optimal solutions for all the literature problems

We have tried 49 problems out of the standard literature set of 64 problems to test the efficacy of EGAALB Optimal solution reported by EGAALB for all these since assembly line balancing is a NP hard problem, it is not always possible to know whether the solution is optimal or not Thus to check the optimality, the solutions of a EGAALB are compared with Hoffmann's (1992) paper The 15 problems which could not be tried are very easy and almost all the literature heuristics have presented optimal answer for them Most of the heuristic fail on Tonge's (70 task cycle time =176s) and Arcus' (111 task, Cycle time =5755s) problem for which EGAALB has provided optimal answers EGAALB has proved itself to be best on these most critical problems still there is a need test its functioning on the remaining 15 problems for the sake of completeness

Thus, quality wise (optimality), EGAALB is the best amongst the heuristic techniques To judge the goodness of an algorithm, computation time is also an important criterion Hoffmann's (1992) ICE procedure took 60 32 seconds to solve 64 problems on Cray-2 This time excludes input / output times However Johnson's FABLE is reported to solve all these in just 3 16 seconds on an IBM-3090 It is interesting to note that EGAALB has solved the 49 problems in less than 2 seconds (i e 1 76s) on HP-9000 which is far less than many of good heuristics Therefore, it can be said that EGAALB, with a combination of IPG-1 and CROSS-2 has proved to be the best heuristic so far when tested on literature problems It is expected that it will perform in a similar way on larger problems of size greater than 1000 tasks However, a more extensive testing is required to verify the performance of EGAALB on larger problems

Chapter 6

CONCLUSIONS

Numerous methods have been proposed for the solution of the assembly line balancing problem (Type-I) Several exact methods exist that guarantee optimality but these procedures are based on non-polynomial algorithms On the other hand, heuristic methodologies provide solution in polynomial time, but the quality of the solution may not be very good Therefore a need of good and efficient heuristic that can give fairly good (optimal or near optimal) solution in considerably small CPU time, is felt The heuristic algorithm based on genetic ideas, EGAALB has reported optimal results for all the literature problems It gives results superior to other heuristics and that too in lesser CPU time

EGAALB is better than the well known and so considered the best heuristic Hoffmann's heuristics because of the following reasons - Hoffmann's heuristic tries all the combinations possible to land up with the first station configuration Then it moves to the next station and so on till all the tasks are assigned In this way, his heuristic tries to push the slack to the last station In backward balancing, the slack is tried to accumulate in the first station This results in optimal balances many times, but not always Actually, in some cases, the optimal balance is present when the maximum slack is present in some intermediate station, other than 1st and last station EGAALB is more successful since it has overcome this negative feature present in Hoffmann's heuristics Several control parameters have also been provided through which the working power of the algorithm can be regulated

A faster varıant of EGAALB ıs when the populatıon sıze = 10 and the number of generatıons = 1 Adoptıng small values of N , n_{gen} and $n_{trial\text{-}set}$ wıll take lesser CPU tıme, and as a tradeoff the qualıty of the solutıon wıll be affected but not to a sıgnıficant extent For example the Arcus' 111 task problem wıll have 111! / 2^{180} dıstınct feasıble sequences where 180 ıs the number of precedence constraınts ın the network An exact method ıs lıkely to evaluate all the possıble combınatıons, but the faster varıant of EGAALB (wıth, $n_{gen} = 1$, $n_{trial\text{-}set} = 10$ and $N = 50$) does only 50 2 10 30 = 1000 evaluatıons to get the optımal answer (Note 30 ıs roughly the number of statıons requıred)

EGAALB has been developed to solve type-1 assembly lıne balancıng problem but ıt can be used repeatedly to solve type-2 problem where the goal ıs to mınımıze the cycle tıme for a gıven number of statıons

The algorıthm developed here ıs meant for sımple assembly lıne balancıng However the logıc of EGAALB can also be applıed to the generalızed assembly lıne balancıng problem whıch ıncludes several complıcatıons vız zonıng contraınt, mıx model lınes etc

REFERENCES

ANDERSON, E J , and M C FERRIS 1994 Genetic algorithms for combinatorial optimization The assembly line balancing problem *ORSA Journal on Computing* **6**, 161-173

ARCUS, A L 1963 An analysis of computer method of sequencing assembly line operations Ph D dissertation, University of California, Berkeley

ARCUS, A L 1966 COMSOAL A computer method of sequencing operations for assembly lines In *Readings in Production and Operations Management*, edited by E S Buffa, Wiley, New York

BAGLEY, J D 1967 The behaviour of adaptive systems which employs genetic algorithms and correlation algorithms Ph D thesis, University of Michigan

BALAS, E 1980 An algorithm for large 0-1 knapsack problems *Operations Research* **28**, 1130-1154

BAYBARS, I 1986 An efficient heuristic method for the simple assembly line balancing problem *International Journal of Production Research* **24**, 149-166

BAYBARS, I 1985 A survey of exact algorithms for the simple assembly line balancing problem *Management Science* **32**, 909-932

BOWMAN, E H 1960 Assembly line balancing problem by linear programming *Operations Research* **8**, 385-389

CHARLTON, J M , and C C DEATH 1969 A general method for machine scheduling *International Journal of Production Research* **7**, 207-217

CONWAY, R W , W L MAXWELL, and L W MILLER 1967 Theory of scheduling Reading Mass Addison Wesley

DAR-EL, E M , and R F GOTHER 1975 Assembly lıne sequencıng for model mıx *Inteınatıonal Journal of Productıon Research* **13,** 463-467

DAR-EL, E M 1975 Sıngle model assembly lıne balancıng problems a comparatıve study *A I I E Transactıons* **5,** 164-171

DAR-EL, E M , and E M MANSOOR 1973 A heurıstıc technıque for balancıng large sıngle model assembly lınes *A I I E Transactıons* **5,** 343-356

DAR-EL, E M , and Y RUBINOVITCH 1979 "MUST" - A multıple solutıon technıque for balancıng sıngle model assembly lınes *Management Scıence* **25,** 1105-1115

DAVIS, L 1991 *The Handbook of Genetıc Algorıthm,* edıted by L Davıs, Van Nostrand Reınhold, New-York

FREEMAN, D R 1968 A general lıne balancıng model *Proceedıngs of the 19th annual conference AIIE,* Tampa, FLA

GAREY, M B , and D S JOHNSON 1981 Approxımate algorıthms for the bın packıng problem a survey In *Analysıs and desıgn of algorıthms ın combınatorıal optımızatıon*, edıted by G Auselıo and M Lucertını, New York

GUTJAHR, A L , and G L NEMHAUSER 1964 An algorıthm for lıne balancıng problem *Management Scıence* **11,** 308-315

HACKMAN, S T , M J MAGAZINE, and T S WEE 1989 Fast and effectıve algorıthms for sımple assembly lıne balancıng problems *Operatıons Research* **37,** 916-9

HELD, M , and R M KARP 1962 A dynamıc programmıng approach to sequencıng problems *Jc urnal of Soc and Applıed Mathematıcs* **10,** 196-210

HELD, M , R I KARP, and R SHARESIAN 1963 Assembly lıne balancıng problem - d namıc programmıng wıth precedence constraınts *Operatıons Research* **1** 4? -459

HELGESON and D B BIRNIE 1961 Assembly lıne balancıng usıng ranked posıtıonal weıght technıques *Journal of Industrıal Engıneerıng* **12,** 394-398

HOFFMANN, T R 1963 Assembly lıne balancıng wıth a precedence matrıx *Management Scıence* **23,** 551-562

HOFFMANN, T R 1992 Eureka A hybrıd system for assembly lıne balancıng *Management Scıence* **38,** 39-47

JACKSON, J R 1956 A computıng procedure for lıne balancıng problem *Management Scıence* **2,** 261-271

JACKSON, J R 1979 A computıng procedure for a lıne balancıng problem *Management Scıence* **25,** 1105-1115

JOHNSON, R V 1973 Branch and bound algorıthms for assembly lıne balancıng problem and job shop schedulıng Ph D dıssertatıon, Unıversıty of Calıfornıa, Los Angeles

JOHNSON, R V 1981 Assembly lıne balancıng computatıonal comparısons *Internatıonal Journal of Productıon Research* **19,** 277-284

JOHNSON, R V 1988 Optımally balancıng large assembly lınes wıth 'FABLE' *Management Scıence* **34,** 240-253

KARP, R M 1972 Reducıbılıty among combınatorıal problems In *Complexıty of Computer Applıcatıons*, edıted by R E Mıller and J W Thatcher, Plenum, NY

KOTTAS, J F , and H S LAU 1973 A cost orıented approach to stochastıc lıne balancıng *A I I E Transactıons* **5,** 164-172

KOTTAS, J F , and H S LAU 1976 A total operatıng cost model for paced lınes wıth stochastıc task tımes *A I I E Transactıons* **8,** 324-331

KOTTAS, J F , and H S LAU 1981 A stochastic line balancing procedure *International Journal of Production Research* **19,** 177-193

MACASKILL, J L C 1972 Production line balances for mixed model lines *Management Science* **19,** 423-434

MAGAZINE, M S , and T S WEE 1981 An efficient branch and bound algorithm for assembly line balancing problem - Part I and Part II Working paper, Department of Management Science, University of Waterloo

MASTOR, A A 1970 An experimental investigation and evaluation of production line balancing techniques *Management Science* **16,** 728-735

MOODIE, C L , and H H YOUNG 1965 A heuristic method of assembly line balancing for assumptions of constant or variable work element times *Journal of Industrial Engineering* **16,** 23-33

REEVE, N R , and W H THOMAS 1973 Balancing stochastic assembly lines *A I I E Transactions* **5,** 223-232

SALVESON, M E 1955 The assembly line balancing problem *Journal of Industrial Engineering* **6,** 18-31

TALBOT, F B , J H PATTERSON, and, W V GEHRLEIN A comparative evaluation of heuristic line balancing technique *Management Science* **32,** 430-454

THOMOPOULOS, N 1970 Mixed model line balancing with smoothed station assignments *Management Science* **16,** 321-335

IME-1995-M-G